DOCUMENT RESUME

ED 116 658 IR 002 900

AUTHOR	Gerlach, Vernon S.; And Others
TITLE	Algorithms in Learning, Teaching, and Instructional Design. Studies in Systematic Instruction and Training Technical Report 51201.
INSTITUTION	Arizona State Univ., Tempe. Coll. of Education.
SPONS AGENCY	Air Force Office of Scientific Research, Arlington, Va.
PUB DATE	Dec 75
NOTE	70p.
EDRS PRICE	MF-$0.76 HC-$3.32 Plus Postage
DESCRIPTORS	*Algorithms; *Instructional Design; Instructional Systems; Research Problems; *State of the Art Reviews; Teaching Methods

ABSTRACT

An algorithm is defined here as an unambiguous procedure which will always produce the correct result when applied to any problem of a given class of problems. This paper gives an extended discussion of the definition of an algorithm. It also explores in detail the elements of an algorithm, the representation of algorithms in standard prose, flow charts, coded graphs, linear representation, list form and decision table form. It develops a taxonomy of algorithms and discusses at length the uses of algorithms in instruction and in research and development problems. (JY)

****
* Documents acquired by ERIC include many informal unpublished *
* materials not available from other sources. ERIC makes every effort *
* to obtain the best copy available. Nevertheless, items of marginal *
* reproducibility are often encountered and this affects the quality *
* of the microfiche and hardcopy reproductions ERIC makes available *
* via the ERIC Document Reproduction Service (EDRS). EDRS is not *
* responsible for the quality of the original document. Reproductions *
* supplied by EDRS are the best that can be made from the original. *
****

ED116658

Rule Learning and Systematic Inatruction in
Undergraduate Pilot Training

Vernon S. Gerlach, Principal Investigator

ALGORITHMS IN LEARNING, TEACHING, AND INSTRUCTIONAL DESIGN

Vernon S. Gerlach
Robert A. Reiser
Fritz H. Brecke

Technical Report #51201

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
NATIONAL INSTITUTE OF EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

Research sponsored by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under Grant No. AFOSR 75-2900. The United States Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

IR 002 900

College of Education
Arizona State University
Tempe, Arizona

December, 1975

"PERMISSION TO REPRODUCE THIS COPYRIGHTED MATERIAL HAS BEEN GRANTED BY
Vernon Gerlach & The State University of Arizona
TO ERIC AND ORGANIZATIONS OPERATING UNDER AGREEMENTS WITH THE NATIONAL INSTITUTE OF EDUCATION. FURTHER REPRODUCTION OUTSIDE THE ERIC SYSTEM REQUIRES PERMISSION OF THE COPYRIGHT OWNER."

Copyright 1975
Arizona State University
All rights reserved

Contents

		Page
I.	On the Difference between Magic and Algorithms	1
II.	The Definition of Algorithm	3
	The Three Attributes	5
	Attribute 1	5
	Attribute 2	7
	Attribute 3	7
	A Traditional Definition	9
	The Relationship between Resultivity and Replicability	9
	A Revised Definition	10
	The Problem Class, the Result Class and the User Class	11
	Domain	11
	Range	11
	User	12
	Summary	13
III.	The Elements of an Algorithm	15
	Introduction	15
	Operator	19
	Discriminator	19
	Syntactic structure	20
	Summary	20
IV.	The Representation of Algorithms	21
	Standard prose	21
	Flow charts	22
	Coded graphs	23

Page

Linear representation 24

List form 25

Decision table form 26

Summary 27

V. Taxonomies of Algorithms 29

VI. The Uses of Algorithms in Instruction 39

Algorithms as Aids to the Learner 39

Algorithms as Aids to the Instructional Designer 41

Algorithms and objectives 41

Entry skills and learning 41

Prompting 44

Individualized instruction 45

VII. Research and Development Problems 53

Algorithms for Learning and Teaching 53

Algorithms in Flying Training 54

Quasi-algorithmic prescriptions and quasi-algorithms 54

Variables in algorithms 56

Syntactic variables 56

Semantic variables 58

Pragmatic variables 58

Non-textual algorithms 59

Epilogue 61

References 63

Appendix 65

List of Figures

Figure		Page
1.	Algorithm for Adding Fractions	4
2.	Euclidean Algorithm (Version 2)	17
3.	Example of an Identification Algorithm in Biology	30
4.	Identification Algorithm after Landa (1974), p. 437	32
5.	Algorithm for Forming the Possessive of English Nouns	33
6.	Algorithm for Adding Fractions	34
7.	Decision Tree for Selecting Evaluation Models	40
8.	Algorithm for Adding Fractions (Version 1)	42
9.	Algorithm for Producing a 3x3 Magic Square	46
10.	Abbreviated Flow Chart	47
11.	A Reminder	48
12.	Euclidean Algorithm (Version 1)	49
13.	Euclidean Algorithm (Version 2)	50
14.	Euclidean Algorithm (Version 3)	51
15.	The Syntactical Structure of the Euclidean Algorithm (Version 1)	57

Acknowledgments

The authors are indebted to Drs. James Eubanks and Robert Haygood and to Maryann Barron for careful reading and critiquing of the paper from first to final drafts; to Diane Stone for handling the typing, editing, reproduction, and a myriad of other details with amazing cheerfulness and dispatch.

To each of them--thank you!

Above all, we take this occasion to thank Dr. William J. Burke, Arizona State's Vice-president for Research from 1962-1975, for his dedication and devotion to the cause of university research and for his encouragement and support during this and all our previous research efforts.

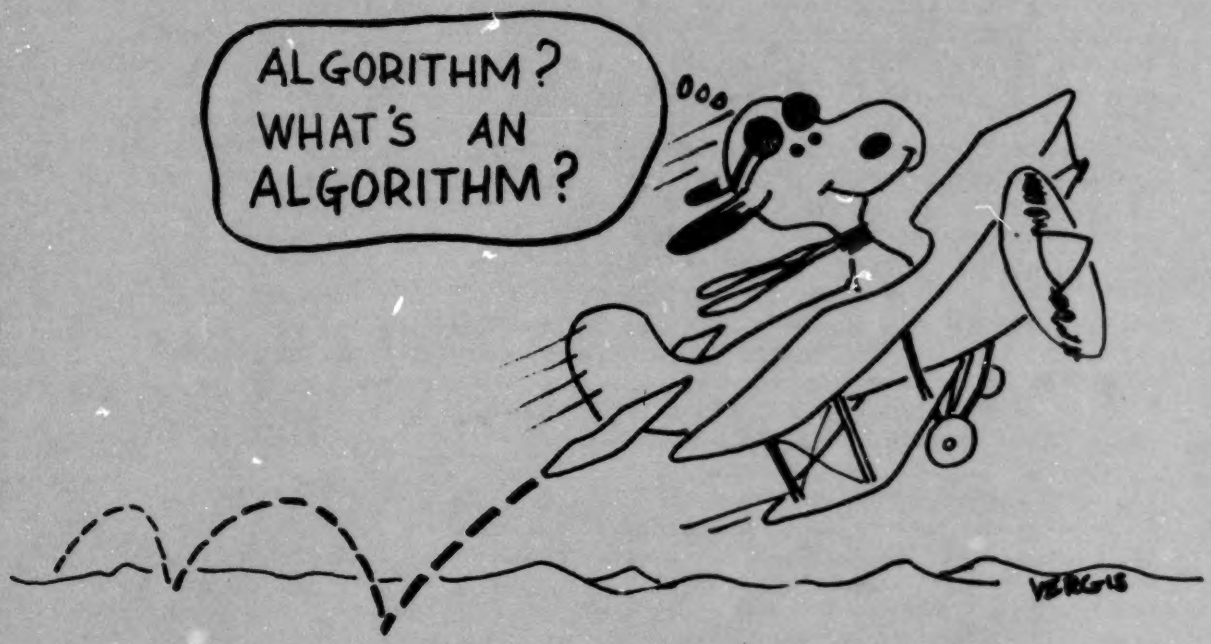
Vernon S. Gerlach
Robert A. Reiser
Fritz H. Brecke

I. On the Difference between Magic and Algorithms

"One day Houdini attended the public demonstration of a French-made flying machine, a Voisin, a beautiful biplane with boxed wings, a box rudder and three delicately strutted bicycle wheels. The aviator flew it over a race track and landed on the infield, and the next day his feat was described in the newspapers. Houdini moved decisively. Within a week he was the owner of a new Voisin biplane. It had cost him five thousand dollars. It came complete with a French mechanic who gave instruction in the art of flying. He secured the use of an army parade grounds outside of Hamburg. In all the countries in which he played he always got on well with the military. Soldiers everywhere were fans of his. Each morning at dawn he would drive to the parade grounds and sit at the controls of the Voisin while the French mechanic lectured him on the function and purpose of the levers and pedals within reach of the pilot. The plane was directed by means of a large steering wheel mounted in the vertical position and attached by a shaft to the front rudder. The pilot sat behind the front rudder on a little seat between the two wings. Behind him was the engine, and behind the engine was the propeller. The Voisin was made of wood. The wings were covered in fabric stretched taut and sized with varnish. The struts connecting the double wings were paneled with the same material. The Voisin looked like a box kite. Houdini had his name painted in block letters on the outside panels of the wings and on the rear elevators. He could hardly wait for his first flight. The patient mechanic drilled him in the various operations required to get the machine aloft, maintain it in flight and land it. Every night Houdini did his act and every morning at dawn he went out for his lessons. Finally one morning when the red sky was clear and the mechanic judged the wind conditions to be right, they pushed the machine out of its shed and faced it into the breeze. Houdini climbed into the pilot's seat, turned his cap backwards and pulled it down tight. He clutched the wheel. His eyes narrowed in concentration, he set his jaw firmly and he turned his head and nodded to the mechanic, who spun the wood propeller. The engine fired. It was an Enfield 80-horsepower job, supposedly better than the one the Wrights themselves were using. Hardly daring to breathe, Houdini throtted the engine, idled it, throttled it again. Finally he held up his thumb. The mechanic ducked under the wings and pulled the wheel chocks. The craft slowly moved forward. Houdini breathed faster and faster as the Voisin picked up speed. Soon it was bumping along the ground and he could feel the sensitive wings take on an intelligence of their own, as if a disembodied presence had joined the enterprise. The machine lifted off the ground. He thought he was dreaming. He had to willfully restrain his emotions, commanding himself sternly to keep the wings level, to keep the throttle continuously in touch with the speed of the flight. He was flying! His feet worked the pedals, he clasped the control wheel and gently the rudder in front of him tilted down and the machine climbed the sky. He dared to look down: the earth was fifty feet below him. He no longer heard the ratcheting engine behind his ear. He felt the wind in his face and discovered he was shouting. The guy wires seemed to sing, the great wings above and below him nodded and dipped and played in the air with their incredibly gentle intelligence. The bicycle wheels spun slowly, idly in the breeze. He was flying over a

stand of trees. Gaining confidence he put the craft into a difficult maneuver, a bank. The Voisin described a wide circle around the parade grounds. Then he could see the mechanic standing in the distance, by the shed, raising both arms in salute. Cooly, Houdini leveled the wings, slipped under his breeze and began his descent. The moment the wheels touched down, the crudeness of the impact offended him. And when the machine rolled to a stop he wanted only to be airborne again." (Doctorow, 1975, pp. 84-86.)

Either Houdini was a magician of considerably greater magnitude than any of us has heretofore suspected, or he used an extremely powerful algorithm.[1]



[1]This is not to deny the possible role of other extremely important variables, such as luck.

II. The Definition of Algorithm

A student pilot is about to make his first attempt at flying a Vertical S-A maneuver in a flight simulator. Both his instructor pilot and his manuals have told him that in order to make the transition from straight and level to steady state climb he should:

1. Apply power at a smooth, slow, and steady rate.

2. As soon as the air speed starts to increase, increase the pitch sufficiently to maintain 160 KIAS.

3. Keep increasing both pitch and power while maintaining 160 KIAS until attitude indicator is +1¼ bar width.

4. The tachometer should show 94 $\pm$ 1%.

5. The vertical velocity indicator should be at or approaching 1000'/minute.

6. Fine tune the vertical velocity to 1000'/minute by making very small and smooth corrections.

7. Trim for hands-off condition as soon as possible.

The student pilot has been taught these verbal statements; now he is about to apply them, as a procedure.

In the context of flying training, procedure conveys information about such things as flight parameters--numerical values such as the power, airspeed, vertical velocity, attitude, and heading for a given maneuver. Procedure tells the student pilot what to do. Sometimes procedure is recorded in books, films, tapes; sometimes the student pilot must learn procedure on his own. Procedure is standardized. Every student pilot is required to perform the maneuver according to a specific procedure (or, at least, according to one of a usually very small number of acceptable procedures). But whatever else procedure may be or whatever form it may take, this attribute remains invariate: procedure is an ordered list of instructions or rules.[2]

Elementary and secondary school pupils must learn an enormous number of procedures. Consider the fifth grader as he learns to add fractions. He is taught to use the flow chart in Figure 1.

His first problem is 1/3 + 1/3. He follows the path a A B: Yes, the denominators are the same (a). The sum of the numerators is 2 (A). This sum is placed over the common denominator, 3 (B). The result is

---

[2]The invariants of rules include the following: they are (1) directed to someone and (2) specify with varying degrees of precision how a certain process is to be carried out.

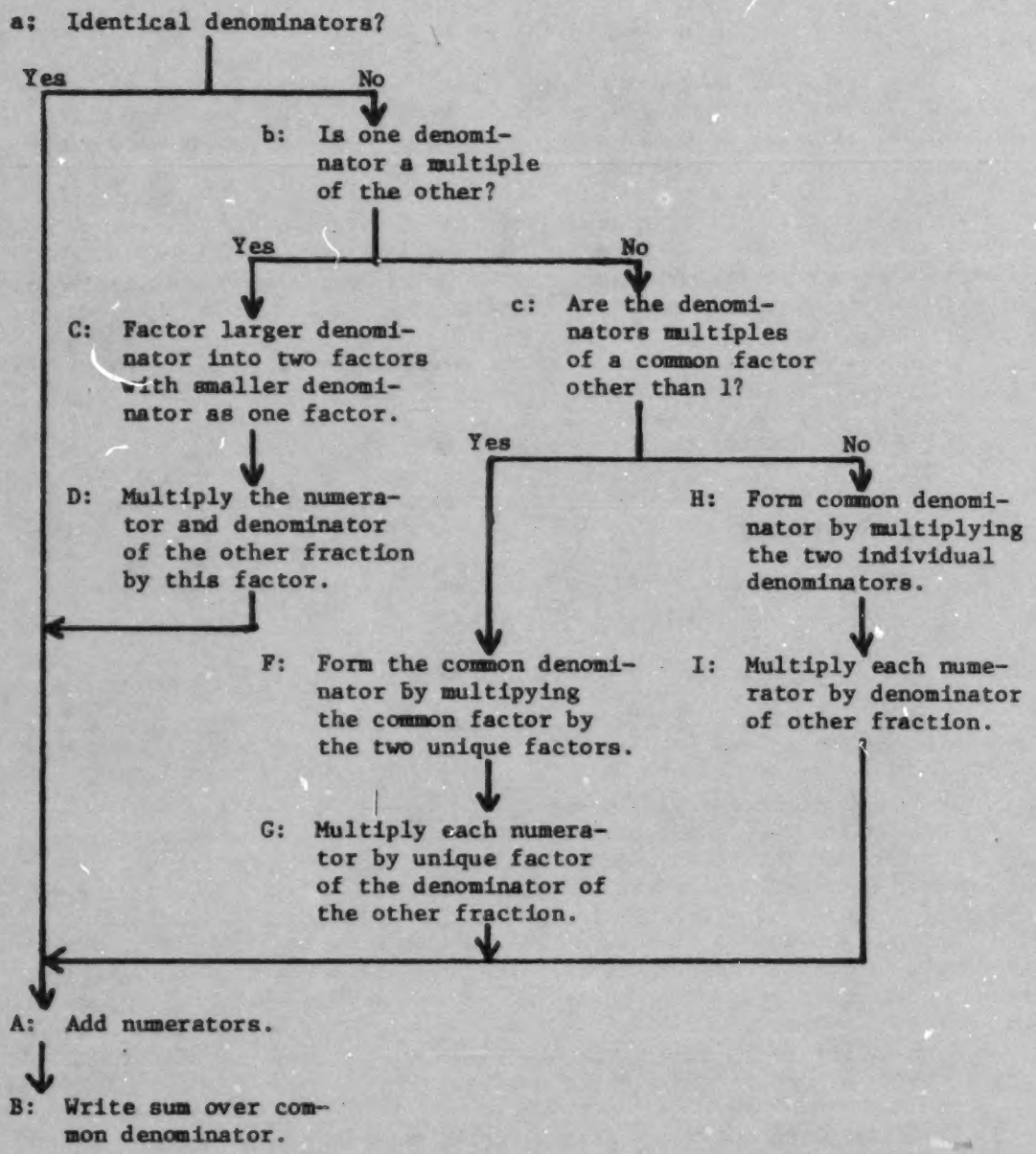

a; Identical denominators?
Yes
No
b: Is one denominator a multiple of the other?
Yes
No
C: Factor larger denominator into two factors with smaller denominator as one factor.
c: Are the denominators multiples of a common factor other than 1?
D: Multiply the numerator and denominator of the other fraction by this factor.
Yes
No
H: Form common denominator by multiplying the two individual denominators.
F: Form the common denominator by multipying the common factor by the two unique factors.
I: Multiply each numerator by denominator of other fraction.
G: Multiply each numerator by unique factor of the denominator of the other fraction.
A: Add numerators.
B: Write sum over common denominator.

Figure 1. Algorithm for Adding Fractions

2/3. Check it for yourself by following path a A B. Several days later he has advanced to a much more difficult problem: 3/16 + 2/3. He follows the path a b c H I A B. Follow the flow chart: (a) the denominators 3 and 16 are not identical; (b) 16 is not a multiple of 3; (c) 3 and 16 are not multiples of a common factor other than 1; (H) the common denominator is 48 (the product of 16 and 3); (I) 3 x 3 = 9 and 2 x 16 = 32; (A) 9 + 32 = 41; (B) 41/48.

What do the student pilot and the fifth grader have in common? Both are following a procedure. The procedure which the fifth grader uses is an algorithm—represented by means of a flow chart. At this point we want to establish only one point: the type of procedure describing how to add fractions is an algorithm. All algorithms are a subset of the set procedure. Every algorithm is an ordered list of instructions or rules, but not every list of instructions or rules is an algorithm.

The procedure for executing the Vertical S-A is not an algorithm; the procedure for adding two fractions is. Why do we make this distinction? Because the procedure for adding fractions possesses three attributes--attributes which every algorithm must possess—but the procedure for flying the Vertical S-A does not. Let us discuss each of these attributes in detail.

The Three Attributes

Attribute 1. Look at the procedure for adding fractions. What problems can be solved by means of this procedure? Is it possible to use the algorithm for any examples other than 1/3 + 1/3 and 3/16 + 2/3? Obviously, yes! It is this characteristic which constitutes the first defining attribute of an algorithm: an algorithm must possess generality; it must be applicable to a class of problems, not merely to a single problem.

The procedure for executing a Vertical S-A does possess generality, but the class of problems to which it is applicable is quite restricted. The procedure cannot be applied to any Vertical S-A maneuver, but can be applied only to Vertical S-A maneuvers in which airspeed is 160 KIAS and altitude range is 1000 feet. The procedure, as defintd by the seven elements on p. 3, is not generalizable to Vertical S-A maneuvers which involve an airspeed other than 160 KIAS or an altitude range other than 1000 feet.

Generally speaking, the literature on algorithms includes generality as one of the defining attributes of the concept; sometimes this attribute is mentioned quite explicitly, sometimes only implicitly. Writers who use the term algorithm with precision are referring to a procedure which may be applied to any problem of a certain class. Trakhtenbrodt (1963) defines the term thus: "By an algorithm is meant a list of instructions specifying a sequence of operations which will give the answer to any problem of a given type [italics added]." Markov (1961) is more general and defines an algorithm as "an exact prescription defining a computational process that leads from various initial data [italics added] to a described result." Even though these formulations are

inadequate as definitions, they do convey the concept of a general procedure for the solution of any problem of a class of problems. Knuth (1968) describes the generality characteristic implicitly when he states that the inputs for an algorithm must be members of a specified set. Landa (1974) uses a definition which includes the attribute generality:

> By algorithm is usually meant a precise, generally comprehensible prescription for carrying out a defined (in each particular case) sequence of elementary operations (from some system of such operations) in order to solve any problems belonging to a certain class (or type). (p. 11)

Bellman, Cooke, and Lockett (1970) agree: ". . . an algorithm . . . must lead to a solution of 'any problem of a given kind,' rather than to one particular problem only" (p. 57). Trakhtenbrodt (1963) uses the lable "generality" as follows:

> The generality of algorithms
>
> An algorithm is a single list of instructions defining a calculation which may be carried out on any initial data and which in each case gives the correct result. In other words, an algorithm tells how to solve not just one particular problem, but a whole class of similar problems. (p. 7)

Horabin and Lewis (1974) have shown that the generality of an algorithm is not dependent on the nature of the subject matter dealt with. There are explicated natural laws in such fields as mechanics, electronics, thermodynamics, or logic. Certain courses deal with the teaching of rules for dealing with systems based on explicated natural laws. Such rules presented in algorithmic form are called grounded rules and the algorithms are called grounded algorithms. In contrast, agreemental rules (Horabin and Lewis, 1974) deal with such matters as tax codes, rules for games and sports, insurance claims, or loan applications. An algorithm for completing a tax form, for example, is not based on any natural rule but on rules formulated by agreement; such an algorithm is called an agreemental algorithm.

The algorithm for adding fractions is grounded. It is based on natural rules for manipulating symbols (in this case, numeric symbols). The rules for adding fractions are not a matter of agreement from time to time; they are "changeless," external, inherent in the arithmetic system.

It is less important to discriminate between grounded and agreemental algorithms than it is to remember that each type must possess generality if it is a true algorithm. An algorithm for completing Tax Form 1040 is agreemental; an algorithm for computing the rate of descent of falling bodies is grounded. However, the former is an algorithm only if it enables the user to complete the forms correctly for the many different instances of income, deductions, and tax liabilities which characterize the population for which Form 1040 is appropriate. Likewise, the rate-of-falling-bodies algorithm is an algorithm only if it is applicable to each and every instance of a class of falling-body problems.

Attribute 2. Look again at the procedure for adding fractions (p. 4). Each step in the procedure is unambiguous, each step fully specifies the action to be taken. Every user who is able to perform each of the steps specified in the procedure will perform these steps in a uniform manner. This characteristic of the procedure for adding fractions brings us to the second defining attribute of an algorithm: an algorithm must possess replicability; it must specify an unambiguous procedure.

The procedure for executing a Vertical S-A does not possess replicability; some of the steps in the procedure are not unambiguous. For example, it is quite unlikely that every pilot who has the necessary entry skills will perform the first step in the procedure ("apply power at a smooth, slow, and steady rate") in the same manner.

Landa (1974), in his section on the definition of algorithms, uses the term specificity in a manner similar to the way we use the term replicability. He says:

> This property specificity] resides in the requirement that the prescriptive directions in algorithms must be strictly defined. Directive instructions must indicate precisely the nature and conditions of each action, exclude chance components in the choice of actions, be uniformly interpretable, and be unambiguous. Thus, they must refer to sufficiently elementary operations for an addressed system--person or a machine--to carry them out unequivocally.
>
> The specificity of an algorithm is expressed in the fact that problem solving by algorithm is a strictly directed process, completely guided and not admitting of any arbitrariness. This is a process which can be repeated by any person (or machine, if the algorithm is programmed into it) and will lead to identical results, if the two data sets are identical. (p. 17)

Trakhtenbrodt (1963) also discusses replicability, although he does not use that term: "An algorithm must be given in the form of a finite list of instructions giving the exact procedure to be followed at each step of the calculation." Bellman et al. (1970) also indicate that an algorithm must possess replicability. They state that an algorithm "specifies the exact procedure to be followed at each step."

Attribute 3. Consider the procedure for adding fractions once more (p. 4). No matter how many times a user with the necessary entry skills performs the procedure to find the sum of two fractions, he will always obtain the correct result. This characteristic of the procedure for adding fractions leads us to the third defining attribute of an algorithm: an algorithm must possess resultivity; it must always lead to a correct result.[3]

---

[3] Of course, when a user performs any procedure, algorithmic or otherwise, there is a chance that he will commit an error. However, in order

The literatue on algorithms is more consistent with respect to resultivity than it is with respect to any other attribute. The writers whom we have read either state or imply that an algorithm must possess the attribute resultivity. Trakhtenbrodt (1963) refers to a sequence of operations which will give the answer to any problem of a class. Markov (1961) states that an algorithm must lead to a described result. Knuth (1968) lists as one of five characteristics of an algorithm the requirement that it produce the correct result. Bellman et al. (1970) imply the same when they assert that an algorithm must lead to a solution of any problem of a given kind.

Since the term resultivity appeared first in the English translation of Landa (1974), and since Landa's definition does have a minor weakness, we ought to examine it carefully. He says:

> Resultivity. This property is reflected in the fact that an algorithm always converges on a specific sought-for result, which is always obtained in the presence of the appropriate data set. This property of an algorithm, however, does not assume that algorithms result in the obtaining of the desired result with all data sets belonging to the defined class. It is possible that the algorithm will be inapplicable to certain sets of data; and, in that case, the process of carrying out the algorithm will either halt suddenly, or it will never end. (p. 18)

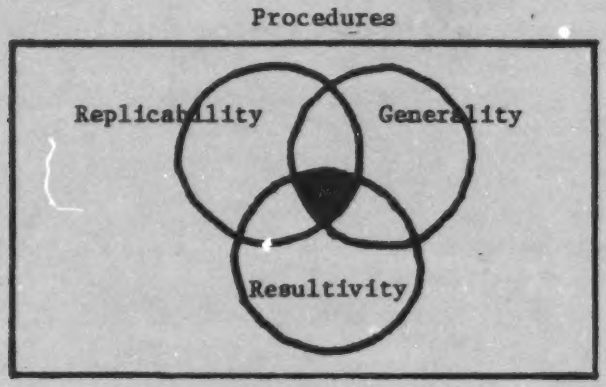
The weakness of this explanation lies in the warning that this property does not always apply, i.e., that it depends on the data set to be processed. This particular difficulty could be resolved by specifying that sets of data which are unsolvable by an algorithm do not belong to the "defined class" or, conversely, that the class must be defined so that the algorithm is applicable to all members of the class. Thus, resultivity becomes a property which is as unconditional as generality or replicability.

---

for a procedure to be considered an algorithm, errors must be attributable to the user and/or factors in the user's environment, rather than to the algorithm itself. For example, if a user made a computational error while performing the procedure for adding fractions, this error could be attributed to the user and the procedure would still be considered an algorithm. Such errors are called ambient errors; they are a function of human vagaries; they wander in and out, unpredictably. However, if one of the steps in the procedure for adding fractions were revised, and this revision led users to add fractions incorrectly, then the errors could be attributed to the procedure and the procedure would not be considered an algorithm. Errors of this type are systematic. If we can do so without provoking an argument on the subject of determinism, we would like to state that systematic errors can be predicted and controlled.

A Traditional Definition

We have now described the three attributes that a procedure must possess in order for the procedure to be considered an algorithm. When these three attributes are combined into one statement, the following tentative definition of an algorithm emerges: an algorithm is an unambiguous procedure which will always produce the correct result when applied to any problem of a given class of problems. In other words, an algorithm is a procedure which possesses replicability, resultivity, and generality. The Venn diagram, below, represents this relationship between the set "procedures" and the subset "algorithms;" the shaded area represents those procedures which are algorithms.

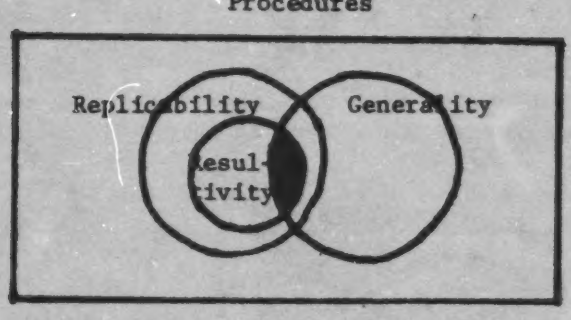


The Relationship Between Resultivity and Replicability

While algorithms have traditionally been defined in terms of the three attributes discussed, algorithms can be defined more simply. Let us examine the procedure for executing a Vertical S-A again. We have already stated that this procedure does possess generality, but not replicability. Neither does it possess resultivity, since not every pilot who has the necessary entry skills will perform the Vertical S-A correctly by following the specified procedure. The unlikelihood of this occurrence appears to be attributable to the ambiguous nature of some of the steps in the procedure for executing the Vertical S-A. If this is true, we can assert that the procedure for executing the Vertical S-A is not resultive because it is not replicable.

Upon further examination, one can say that any procedure which is not replicable will not be resultive. That is, if a procedure is not specified in such a way that it will be performed uniformly by all those users who have the necessary entry skills, then it is likely that the lack of uniform performance will, in some instances, lead to the attainment of an incorrect result. Conversely, in order for a procedure to always lead to a correct result, that procedure must be unambiguous. In other words, if a procedure is resultive, it must be replicable. However, one should not assume that resultivity and replicability are synonymous. An incorrect procedure may consistently produce a single incorrect result. Thus, an incorrect procedure can be replicable, although it certainly would not be resultive. In summary, all resultive procedures are

replicable, but all replicable procedures are not resultive; in other words, resultivity is a proper subset of replicability. A revised version of our Venn diagram, based on the relationship between resultivity and replicability, is presented below; the shaded area represents procedures which are algorithms.



A Revised Definition

The Venn diagram shows us clearly that algorithms can be defined as those procedures which possess resultivity and generality. In other words, an algorithm is a procedure which will produce the correct result when applied to any problem of a given class of problems. It is not necessary to specify that an algorithm must possess replicability (be unambiguous) because, as we have pointed out, resultivity imples replicability.

This definition is a significant departure from those presented in the literature as discussed above. It is particularly significant that Landa (1974), who lists generality, resultivity, and specificity as essential attributes, actually uses the term specifity (which is synonymous with replicability) in much the same manner that we do. To put it negatively, we cannot find any instance in which Landa uses the term resultivity except with the implication that it is a subset of replicability. Thus, our perception of the literature as well as our own analysis of the concept have led us to the conclusion that replicability is unnecessary and should be excluded from the list of defining attributes. The law of parsimony is sufficient grounds, in our estimation, for reducing the list of defining attributes to two: generality and resultivity.

We emphasize, however, that this simplification is done on purely logical grounds. As yet, we have no evidence on which to base an assertion that there is a pragmatic advantage in eliminating replicability from the definition. Perhaps the simpler definition will facilitate the empirical classification of procedures as either algorithms or nonalgorithms. Determining, on an empirical basis, the unambiguity of a procedure appears to be a formidable task. Our revised definition of algorithms should eliminate the need to conduct such a task.

The Problem Class, the Result Class, and the User Class

Before proceding to a formal analysis of the elements of an algorithm, we want to translate the abstractions of our definition into some relatively concrete referents. We shall do this by considering three descriptors which will always be included in any algorithm presented in subsequent sections.

Domain. We have established that an algorithm must possess generality; it is a procedure which is applicable to any problem of a class. The domain of the algorithm (Bung, 1971) is the entire class of problems for which it will work. In order to establish the generality of any algorithm, the domain must be clearly and explicitly stated.

Look at the fraction adding algorithm once again (p. 4). This algorithm is not for use with problems which consist of three or more addends. Instead, this algorithm should be applied only to sets of two fractions expressed in the conventional numerator over denominator form. It could be restricted to fractions whose terms are natural numbers, but this is not necessary; the algorithm will yield a solution to problems in which one, or several, or all of the terms are literal numbers. You can demonstrate this to yourself by using the algorithm to add $a/x + b/y$, thus:

H: $x \cdot y = xy$

I: $a \cdot y = ay \qquad b \cdot x = bx$

A: $= ay + bx$

B: $\frac{ay + bx}{xy}$

The class of problems, then, to which this algorithm applies is "any set of two fractions expressed in numerator over denominator form." That is the domain of the algorithm. The domain of any algorithm should always be explicitly stated.

Range. The application of an algorithm always leads to a specific correct result which is a member of a set of possible correct results or outputs. This set of possible correct results is called the range of an algorithm (Bung, 1971). In order to establish the resultivity of any algorithm, the range must be clearly and explicitly stated.

In the fraction addition algorithm, the only possible correct answers are numbers which represent sums. The difference between, the product of, or the quotient of two fractions simply will not do. The algorithm must yield a sum. Furthermore, the algorithm will yield a sum only when applied to a pair of fractions; it is not for use with whole numbers, with exponents, with decimals. The range, then, is "the sum of any set of two fractions within the domain." As is the case with the domain, the range should always be presented, irrespective of the form in which the algorithm appears.

User. Every algorithm is applicable to a system. The system may be a human, as in the case of a fifth grader learning to add fractions, or the system may be a machine, as in the case of a key punch being controlled by an algorithm punched into a drum card. In any event, the user (the system for which the algorithm is intended) must possess the capability of using the algorithm.

Theoretically, it could be argued that every algorithm must be represented in a language which can be "understood" by any user and that all formulations in that language are completely unambiguous for any user. Practically, however, it is probably impossible to find or to construct such a language; it is equally impossible to generate formulations which are completely unambiguous for an unlimited range of users.

Let us elaborate this last point briefly. The fraction algorithm possesses generality and resultivity for a fifth grader as well as for a Ph.D. in mathematics. If the latter had forgotten how to add two common fractions, he could find their sum by applying the algorithm to the pair. This algorithm could also be represented by the use of abstract algebraic symbols. If it were, it would still be exactly the same algorithm. However, a fifth grader would be unable to use it.

Because of the variability of potential users of an algorithm, it is essential that a match be found between the algorithm and the users for whom it is intended. The kinds of users for which a given procedure is an algorithm must be specified. In the case of the fraction addition algorithm, the user must have the ability to factor whole numbers. Anyone who has mastered this skill (which, too, may be described by an algorithm) is assumed to have the ability to add and multiply whole numbers. If one cannot make this latter assumption, then either it must be stated explicitly or the algorithm must be revised in one of two ways: the user must be shown how to add and multiply whole numbers or a method of adding fractions must be devised which is independent of the user's ability to add and multiply. Theoretically (certainly not practically!) the latter might be accomplished by means of several tables.

We want to avoid the problem of an infinite regress when we decide on the starting point for an algorithm. No matter what the first step in an algorithm may be, there is always something antecedent which the user must know or be able to do. The same could be said of that antecedent, and so on, and so on, ad infinitum--or at least until one arrives (after several lifetimes of analysis) at that pristine elemental bit of information which is the genesis of everything and anything. Arbitrarily specifying an antecedent knowledge or skill as a starting point, therefore, is a simple means of avoiding the infinite regress.

An algorithm, then, is not complete until an explicit statement of user entry skills is included. We have adopted the convention of listing the skill(s) under the heading entry skills. The statement of entry skills must answer the question, "What must the user know or be able to do in order to use this algorithm?"

One might argue that this descriptor, which is nothing more than the instantialization of the replicability attribute, is either the most

important attribute, or at least equal in importance to generality and resultivity. We readily grant this point. However, since no algorithm can possess replicability if it lacks resultivity, there is no need to insist on the traditional three-part definition. Remember, the user does not define the algorithm; rather, the algorithm defines the user.

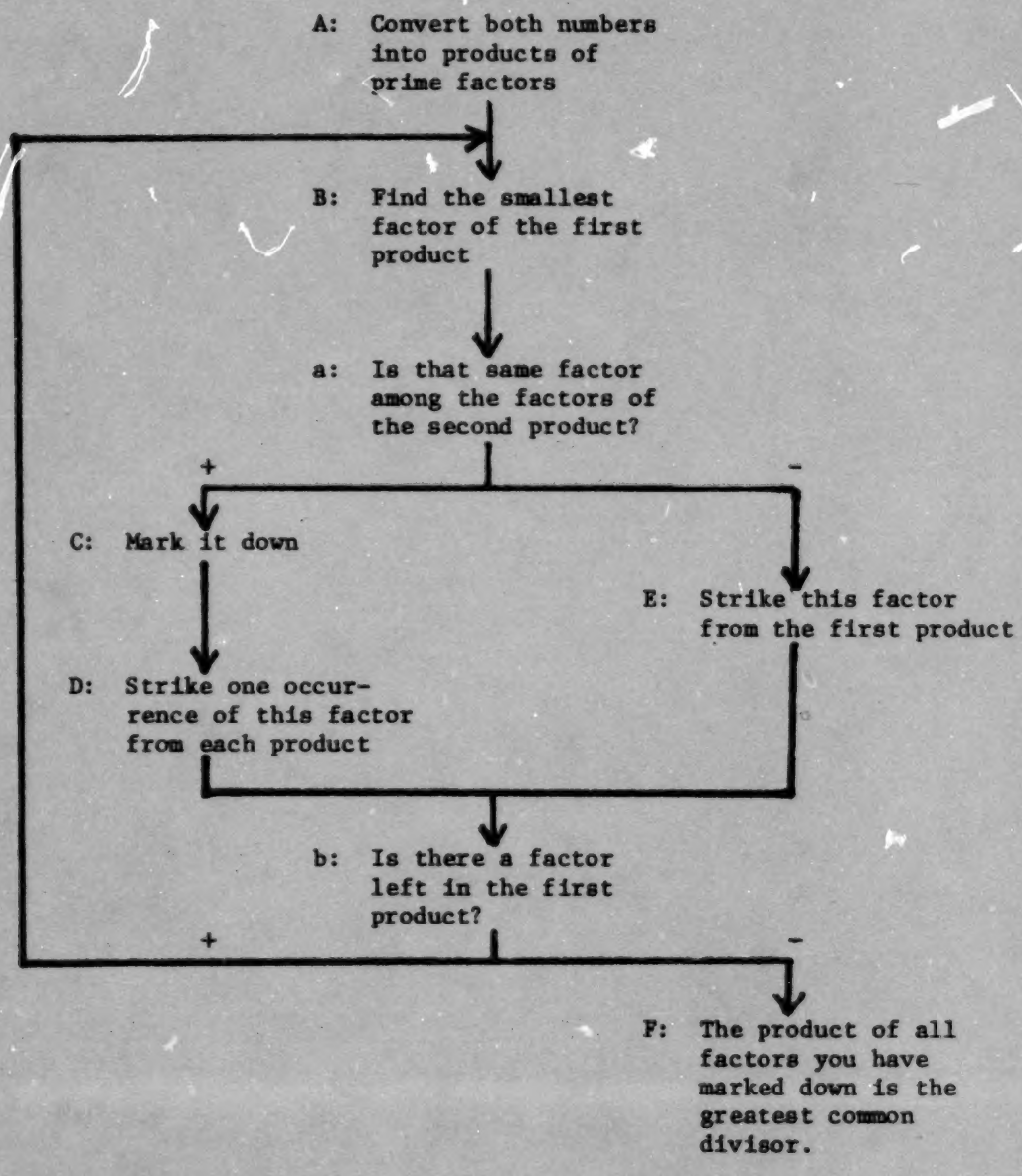
Summary. Every algorithm must be applicable to a class of problems as opposed to a single problem. The class of problems is defined under the domain descriptor. Every algorithm must yield a specific sought-for result which is a member of a set of results. The set of results is defined under the range descriptor. Finally, the user's prerequisite knowledge or skill is specified under the entry skill descriptor.

III. The Elements of An Algorithms

Introduction

This is the classic Euclidean algorithm:

Domain: Any set of two natural numbers
Range: The greatest common divisor for any set of the domain
Entry skill: Factor natural numbers


A: Convert both numbers into products of prime factors
B: Find the smallest factor of the first product
a: Is that same factor among the factors of the second product?
+
-
C: Mark it down
E: Strike this factor from the first product
D: Strike one occurrence of this factor from each product
b: Is there a factor left in the first product?
+
-
F: The product of all factors you have marked down is the greatest common divisor.

The purpose of the Euclidean algorithm is to find the greatest common divisor of two real numbers. For illustration, let us apply the algorithm by using 8 and 12.

A: $8 = 2 \times 2 \times 2$

$12 = 2 \times 2 \times 3$

B: 2

a: Yes

C: 2

D: $8 = \not{2} \times 2 \times 2$

$12 = \not{2} \times 2 \times 3$

b: Yes

B: 2

a: Yes

C: 2

D: $8 = \not{2} \times \not{2} \times 2$

$12 = \not{2} \times \not{2} \times 3$

b: Yes

B: 2

a: No

E: $8 = \not{2} \times \not{2} \times \not{2}$

b: No

F: $2 \times 2 = 4$ (i.e., C x C = 2 x 2)

Exactly the same results will be obtained if the algorithm in Figure 2 is used. Use the same two numbers, 8 and 12, to test the algorithm.

a: Yes, $8 < 12$

A: $12 - 8 = 4$ (4 becomes the new second number)

a: No (8 is not smaller than 4)

b: No, $8 \neq 4$

Domain: Any set of two natural numbers

Range: The greatest common divisor for any set of the domain

Entry skill: Subtract natural numbers

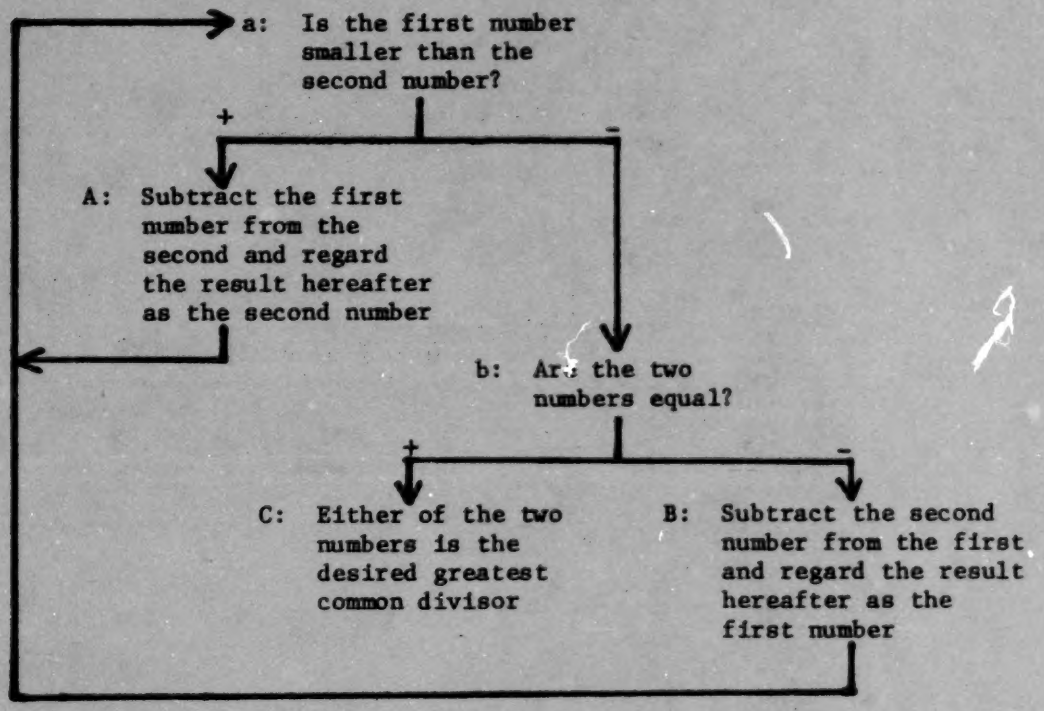

a: Is the first number smaller than the second number?
+
-
A: Subtract the first number from the second and regard the result hereafter as the second number
b: Are the two numbers equal?
+
-
C: Either of the two numbers is the desired greatest common divisor
B: Subtract the second number from the first and regard the result hereafter as the first number

Figure 2. Euclidean Algorithm (Version 2)

B: 8 - 4 = 4

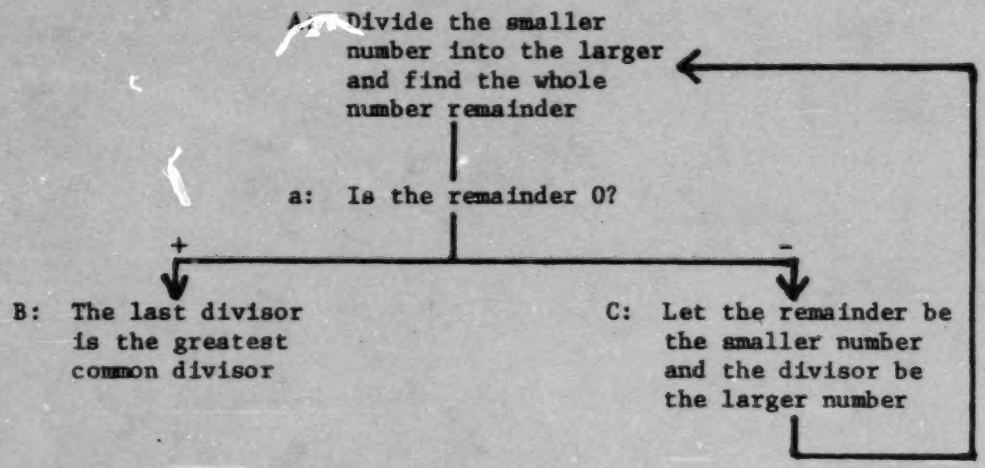
a: No

b: Yes, 4 = 4

C: 4 is the greatest common divisor

This is Shipley's alternative to the Euclidean algorithm:[4]

Domain: Any set of two natural numbers
Range: The greatest common divisor for any set of the domain
Entry skill: Divide whole numbers


A: Divide the smaller number into the larger and find the whole number remainder
a: Is the remainder 0?
+
-
B: The last divisor is the greatest common divisor
C: Let the remainder be the smaller number and the divisor be the larger number

Again, find the greatest common divisor of 8 and 12.

A: 12 ÷ 8 = 1, R 4

a: No

C: Smaller number is 4, larger is 8

A: 8 ÷ 4 = 2, R 0 (no remainder)

a: Yes

B: 4 is the greatest common divisor

---

[4] Adapted from unpublished materials developed by Brian Shipley.

Operator. In each of the three algorithms there are statements which tell you to perform an operation. Statements of this type are called operators and are labeled with upper case letters in our illustrations. Examples of operators include:

B: Find the smallest factor of the first product.

B: Subtract the second number from the first and regard the result hereafter as the first number.

C: Let the remainder be the smaller number and the divisor be the larger number.

The definition of an operator, then, is the type of element in an algorithm which tells the user to perform an operation.

The exit point of the algorithm may not appear to be an operator:

F: The product of all factors you have marked down is the greatest common divisor.

C: Either of the two numbers is the desired greatest common divisor.

B: The last divisor is the greatest common divisor.

At first glance, these statements do not appear to tell the user to perform an operation. Imagine, however, that these operators were changed ever so slightly:

F: Write down the product of all the factors you have marked down; it is the greatest common divisor.

C: Write down either; it is the greatest common divisor.

B: Write down the last divisor; it is the greatest common divisor.

In this augmented form, the statements are obviously operators.

Discriminator. The second type of statement found in each algorithm is that which requires the user to make a decision. Statements of this type are called discriminators and are labeled with lower case letters in our illustrations. Examples of discriminators include:

b: Is there a factor left in the first product?

a: Is the first number smaller than the second?

a: Is the remainder 0?

This is the definition of a discriminator: an element of an algorithm which requires the user to discriminate between two possible conditions or between the presence or absence of a specified condition.

Syntactic structure. The operators and discriminators of an algorithm are related to each other. In the three examples for finding the greatest common divisor, these relationships are represented by means of the lines and arrows as well as by the plus and minus signs. If the algorithms had been represented by some means other than a flow chart, these relationships might have been represented by such statements as "If . . ., then . . .", and "Go to . . .", to mention only the obvious. Regardless of the form, the structure of an algorithm which relates the operators and discriminators is called the syntactic structure.

Syntactic structure is essentially the same concept as Frank's[5] (1969) macrostructure. The notion of a syntactic structure is not peculiar to algorithms. For example, in automata theory this structure (or function) is referred to as the "transformation function" (Glushkow, 1963).

Summary. All algorithms possess operators, discriminators, and a syntactical structure.[6] Operators tell the user of an algorithm to perform an operation. Discriminators require the user to discriminate between two possible conditions or between the presence or absence of a specified condition. The syntactic structure of an algorithm relates the operators and discriminators.

---

[5] Frank defines an algorithm as a triple

$$A = \{R, Y, \Phi\}$$

where:

A represents the set of discriminable attributes or reactions of the object to be controlled.

Y represents the set of operations to be performed on this object.

Φ represents the macrostructural function of the algorithm, i.e., the function which assigns an element of A to each element of Y.

[6] A procedural algorithm may lack any discriminators. For example: A: Turn on the switch. B: Move the clutch lever from "neutral" to "operate" is a procedural algorithm which lacks any explicit discriminators. This paper does not deal with procedural algorithms, since they have extremely limited applicability to learning and instruction.

IV. The Representation of Algorithms

Nearly any algorithm may be represented in a variety of forms which are equivalent in terms of such variables as operators, discriminators, domain of inputs, range of outputs, and required entry skills. The various representational forms differ widely in terms of readability, structural clarity, effort required to produce copy, and space required for publication. The familiar flow chart is usually the clearest and most readable form; unfortunately, it is also the most difficult and time consuming to produce. Other forms of representation include standard prose, coded graphs, linear notational systems, lists, and decision tables. In the following pages an algorithm which we refer to as the "Bird algorithm"[7] is shown in each of these forms.

Standard prose. First, this algorithm is represented in ordinary discursive text. The range is "Leaving; feeding and watering; taking to a veterinarian; burying;" the domain is "Any instance of a person finding a bird lying on the ground;" the entry skill is "Recognizes birds; knows what a veterinarian is." This is the algorithm: "You find a bird lying on the ground. Check if he is still alive. If he is dead, leave him where he is. If he is alive, give 50 ml. of water, and 30 g. of birdseed per day. If he gets better, let him fly. If he doesn't get better and is still alive, take him to a veterinarian and follow his instructions. If he dies, bury him." Note that the complete algorithm must be read if you want to know what to do with the bird. This may not be very taxing in this particular case, since this algorithm is relatively short and concise.

However, the reading of the complete algorithm is a formidable task in the following:

Domain: Price, transaction expenses, and date of purchase of shares of stock; market value on 6 April 1965.
Range: Base for tax allowance or charge.
Entry skill: 7th grade reading ability.

> If the asset consists of stocks or shares which have values quoted on a stock exchange (see also paragraph 6 below), or unit trust units whose values are regularly quoted, the amount of tax chargeable or allowable depends upon the relative sizes of the cost price of the asset, its market value on 6 April 1965, and the selling price of the asset.
>
> If the selling price is greater than the market value, and the market value is greater than the cost price, tax is charged on the selling price less the market value (less allowable expenses). If the selling price is greater than the market value, and the market value is less than the cost price, two possibilities arise. Either the selling price is greater than the cost price, in which case tax is

---

[7] Adapted from unpublished materials developed by Klaus Bung.

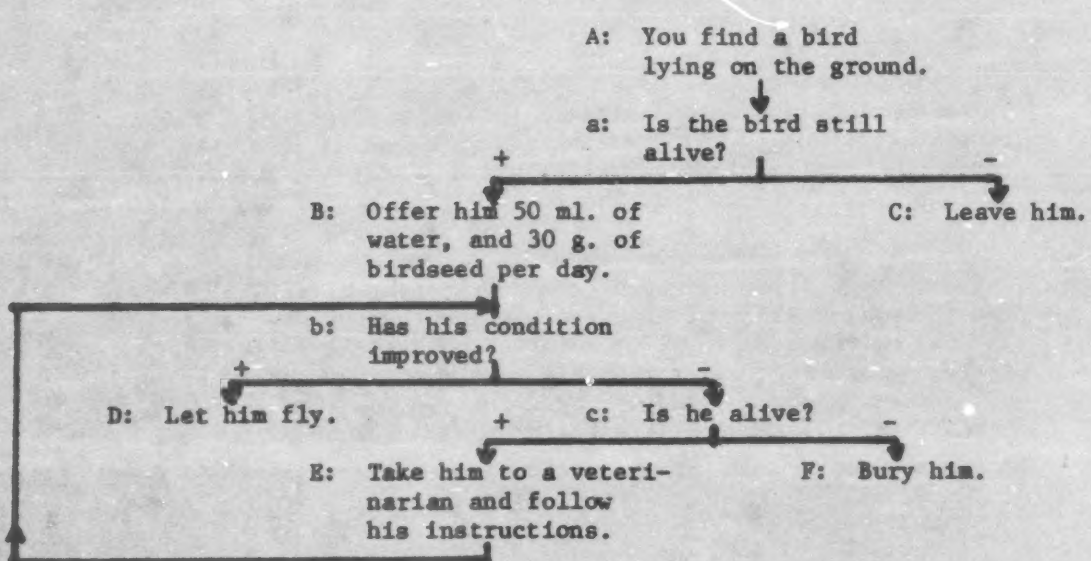
charged on selling price less the cost price (less expenses). Or the selling price is less than the cost price, in which case no tax is either charged or allowed.

If the selling price is less than the market value, and the market value is less than the cost price, tax is allowed on the market value less the selling price (plus allowable expenses). If the selling price is less than the market value, and the market value is greater than the cost price, two possibilities arise. Either the selling price is less than the cost price, in which case tax is allowed on the cost price less the selling price (plus expenses). Or the selling price is greater than the cost price, in which case no tax is either allowed or charged. (Horabin and Lewis, 1974, p. 6)

Even though the version of the tax regulation you have just read is formidable, it is much clearer than the original; nevertheless, it is still difficult to read and even more difficult to apply. The prime reason for the difficulty is that the user is forced to read more than necessary. Assume that the user is computing the tax for one particular instance. Only a few of the many conditions given in the algorithm are applicable to this particular case. Despite this, the user must read everything, trying to discard the irrelevant and remember (or apply) the relevant as he goes along.

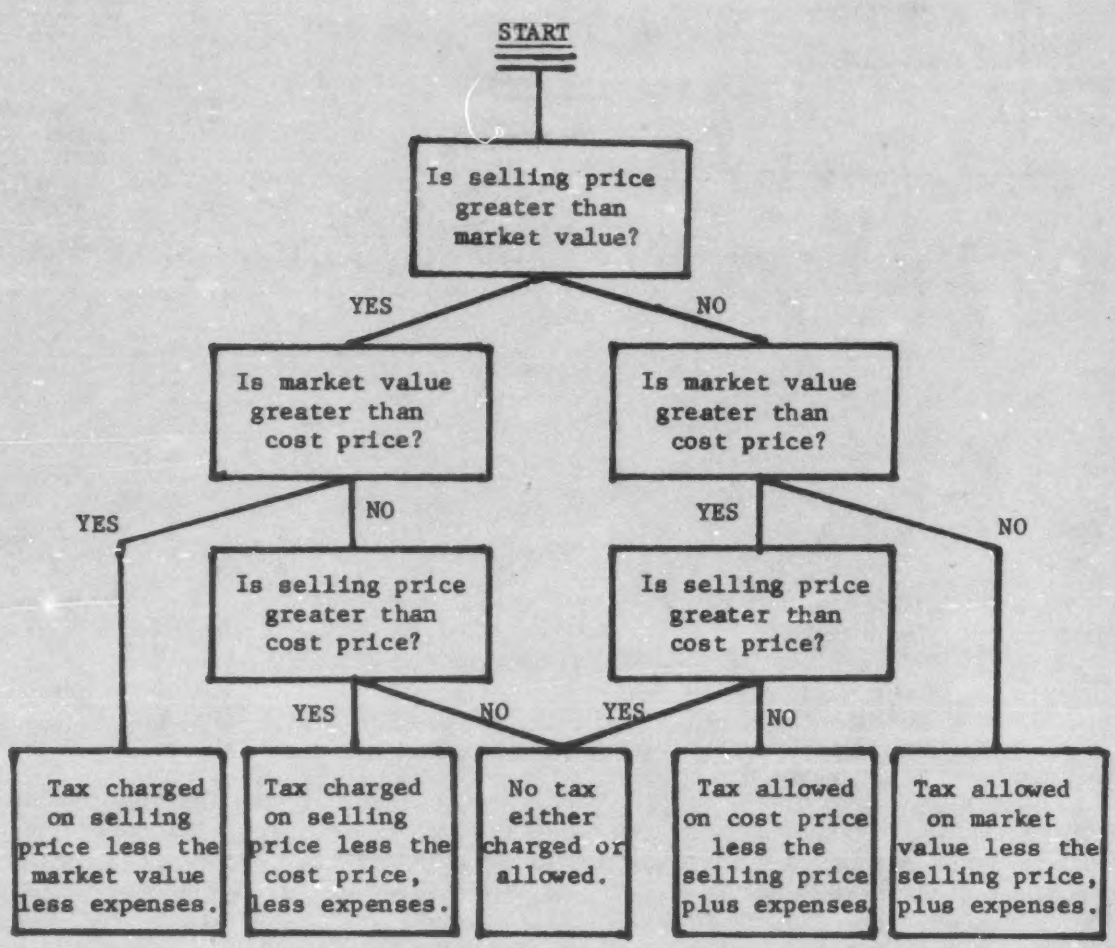
Flow charts. This is the standard form used in the previous sections; its advantages in terms of readability and structural clarity are obvious.

Domain: Any instance of a person finding a bird lying on the ground.
Range: Leaving; feeding and watering; taking to a veterinarian; burying.
Entry skill: Recognizes birds; knows what a veterinarian is.

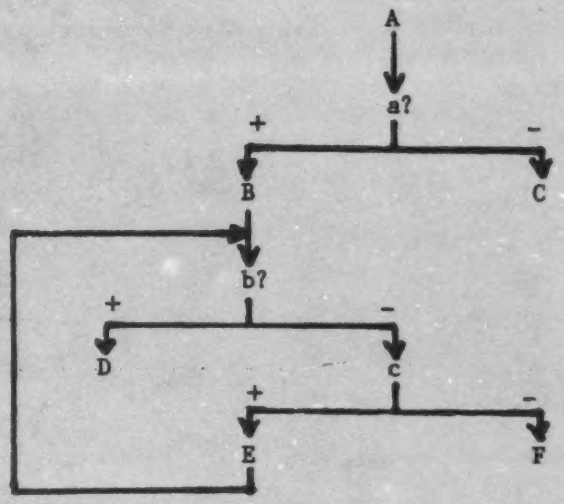

A: You find a bird lying on the ground.
a: Is the bird still alive?
+
-
B: Offer him 50 ml. of water, and 30 g. of birdseed per day.
C: Leave him.
b: Has his condition improved?
+
-
D: Let him fly.
c: Is he alive?
+
-
E: Take him to a veterinarian and follow his instructions.
F: Bury him.

This is the flow chart form for the algorithm for the capital gains tax given above in prose form (Horabin and Lewis, 1974, p. 8):

Domain: Price, transaction expenses, and date of purchase of shares of stock; market value on 6 April 1965.
Range: Base for tax allowance or charge.
Entry skill: 7th grade reading ability.


START
Is selling price greater than market value?
YES
NO
Is market value greater than cost price?
Is market value greater than cost price?
YES
NO
YES
NO
Is selling price greater than cost price?
Is selling price greater than cost price?
YES
NO
YES
NO
Tax charged on selling price less the market value less expenses.
Tax charged on selling price less the cost price, less expenses.
No tax either charged or allowed.
Tax allowed on cost price less the selling price plus expenses.
Tax allowed on market value less the selling price, plus expenses.

Coded graphs. Bung makes a distinction between plain prose graphs such as the one above or coded prose graphs such as the flow chart of the Bird algorithm (see p. 24). In the latter, each operator is preceded by a capital letter and each discriminator is preceded by a lower case letter. (These code letters can also be used to represent algorithms in two other ways, as shown in the next section.)



Coded graphs must be accompanied by a key which associates the code letters with the verbal statements of discriminators and operators. The key for the algorithm above is as follows:

Discriminators

a: Is the bird still alive?
b: Has his condition improved?
c: Is he alive?

Operators

A: You find a bird lying on the ground.
B: Offer him 50 ml. of water, and 30 g. of birdseed per day.
C: Leave him.
D: Let him fly.
E: Take him to a veterinarian and follow his instructions.
F: Bury him.

Linear representation. Longer and more complex algorithms in the form of flow charts take up much space and the typing and drawing is time consuming. These difficulties can be avoided by using a linear form of representation employing arrows and code letters (Lyapunow, 1960). Bung (1969) modified Lyapunow's system so it could be typed with any normal typewriter without the use of arrows. Bung calls this system BULL notation. Both the BULL notation and Lyapunow's system require the use of a key. The Bird algorithm in BULL notation is as follows:

A a 2 B 5 b 3 D. 3 c 4 E 5 y 4 F. 2 C.

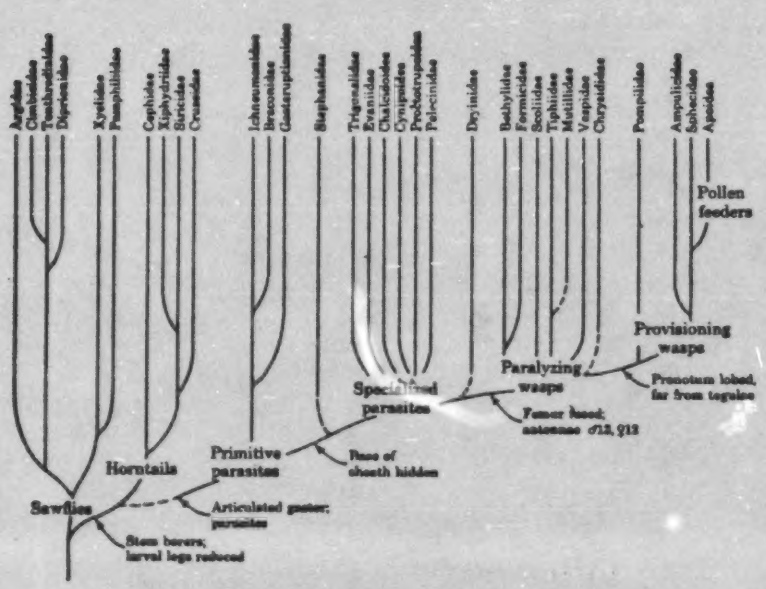
Bung provides the following reading instructions for this notational system:

If the question associated with a discriminator is answered Yes, we call the discriminator positive; otherwise we call the discriminator negative. Operators are denoted by capital letters and discriminators by small letters. The number immediately on the right of a small letter is called 'source number.' Any other number is called 'target number.' The left-most letter is understood to incorporate the start instruction. Read from left to right unless otherwise instructed. After an operator or a positive discriminator, read the nearest letter or full-stop on the right, ignoring any numbers. After a negative discriminator, read the source number adjacent on the right. Then read the identical target number. Then read the letter on the right of the target number. Stop after having read a full-stop.

Bung recommends that the lower case y never be used as a code letter for a discriminator but only to indicate that the number preceding it is a source number for a recursive loop.

List form. The list form is frequently used to represent algorithms for identifying things. Such algorithms are common in botany and zoology; they are frequently called "keys." An example of a key to insect families is the identification tree below:

Domain: All insects of the order Hymenoptera
Range: Each family of the order Hymenoptera
Entry skill: Ability to identify parts of an insect, such as sheath, femur, antennae, pronotum.



The Bird algorithm can be represented in list form:

A: You find a bird lying on the ground.
a: Check whether he is still alive.
If yes, go to B; if no, go to C.
B: Offer him 50 ml. of water, and 30 g. of birdseed per day.
Go to b.
b: Check whether his condition has improved.
If yes, go to D; if no, go to c.
C: Leave him.
D: Let him fly.
c: Check whether he is still alive.
If yes, go to E; if no, go to F.
E: Take him to a veterinarian and follow his instructions.
Go back to b.
F: Bury him.

The first operator and discriminator, A and a, can be contracted into one instruction since the sequence operator→discriminator is unequivocal. This is not the case for operator B and discriminator b. Discriminator b can also be reached via the recursive loop originating from operator E. The list, therefore, has to have separate items B and b in that sequence.

The algorithm in list form shown below (Landa, 1974), represents a very simple algorithm for starting a type of machine.

1. Verify whether the apparatus is plugged in.
If yes, then proceed to instruction 3.
If not, then proceed to instruction 2.
2. Plug it in.
3. Flip the switch.
4. See whether the red light has come on.
If yes, then proceed to instruction 5.
If not, then proceed to instruction 6.
5. Begin work.
6. Call the technician.

Note that both this list form and the flow chart form offer advantages over the plain prose form. The reader has to read only what is relevant to his specific problem when he uses the list or flow chart forms of representation. However, the flow chart form provides a graphic illustration or a picture of the relationships among the elements of the algorithm; the list form does not.

Decision table form. This form is not amenable to every type of algorithm. Possibly only identification algorithms, such as the key on p. 24, can be put into this form. If the algorithm includes transformations which must be observed over time, as is the case in the Bird algorithm, there is no way to list the intermediate operations. Davies (1971, p. 143) gives the example of decision table for an official regulation shown below:

Decision Table for the Death Grant Regulation

CONDITION STUB	CONDITIONS ENTRIES					
Q1 Were the contributions paid late?	No	Yes	Yes	Yes	Yes	Yes
Q2 Were the contributions paid before the death of the subject of the claim?	--	No	Yes	Yes	Yes	Yes
Q3 Is the insured person alive?	--	--	No	No	No	Yes
Q4 Were the contributions paid before the insured person died?	--	--	No	No	Yes	--
Q5 Have the contributions already been taken into account for a claim for a widow's or retirement pension?	--	--	No	Yes	--	--
ACTION STUB	**ACTION ENTRIES**					
Death grant is payable.	*			*	*	*
Death grant is NOT payable.		*	*			
Rules	(1)	(2)	(3)	(4)	(5)	(6)

(A dash in the condition entry column indicates that either a yes or a no answer is acceptable. In other words, the answer to the question does not affect the final outcome.)

Summary. An algorithm may be represented in a variety of ways. Some of the more common forms are flow chart, standard prose, coded graph, linear notational system, list, and decision table. Despite the fact that flow charts require considerable production time and a great deal of page space, they have advantages of clarity and economy for the user.

V. Taxonomies of Algorithms

The algorithm in Figure 3 was constructed to enable zoology students to identify the family to which examples of the order neuroptera belong. Consider a very simple example. You've found an insect with two pairs of clear wings having many veins and crossveins. It has chewing-type mouth parts, long and multisegmented antennae, and large eyes. You're quite certain that it is a member of the order neuroptera; you want to identify the family to which it belongs.

First, a disclaimer: If your hypothesis concerning the name of the order is wrong, the algorithm simply won't function. Earlier it was established that a true algorithm possesses the attribute generality; it will yield a correct result when used for any problem of a certain class. In this case, the class of problems is "identifying the family to which a given specimen of the order neuroptera belongs."

Let's return to the task. The insect (a) has front legs with apical segments slender, same as other legs; (b) it is a small insect with numerous veins and crossveins and it is covered with a waxy bloom; (c) the front wings have a regular, fencelike series of 16 crossveins (graduate veins), similar to those between R_1 and R_s in Figure 285E; (d) the antennae are long and slender, as in Figure 284, tapering to an apex.

Look at the description, just given, and find each characteristic in the algorithm. Characteristic "a" is found in the first member (1) of the algorithm. The critical portion of this statement (the discriminator) is the last part; this branch of the discriminator leads to the "2". (Technically, the discriminator is, "Do the front legs have slender apical segments?" The response yes leads the user to the operator, "Go to 2.")

Use the second member of the algorithm to make the appropriate discrimination concerning characteristic "b" (small; numerous veins, crossveins; waxy). You go to 3.

Characteristic "c" (series of 16 crossveins) is processed by means of the third element. The algorithm forces you to make the discrimination which leads you to 4.

Characteristic "d" (long, slender antennae) meets the requirement of the first discriminator in the fourth element. This is an exit point. You've used the algorithm to complete your task. The result is that you correctly identify the family chrysopidae as the one to which the specimen belongs.

This type algorithm is an identification algorithm, one used to identify an object as belonging to a certain class of objects, events, symbols, or characteristics. The object must be a member of the class (or set) of inputs for which a given algorithm is intended (See "Domain," Section II, above). In the example above, it was pointed out that the algorithm would function only if the specimen was indeed a member of the order neuroptera.

Domain: All insects of the order Neuroptera.
Range: Each family of the order Neuroptera.
Entry skill: Ability to identify parts of an insect: veins, antennae, wings; ability to recognize insects of the order Neuroptera.

Order NEUROPTERA · Lacewings, Mantispids

The adults are minute to large insects, usually with two pairs of clear wings having many veins and crossveins, with chewing type mouthparts, long and multisegmented antennae, and large eyes, fig. 284*A*. The larvae are varied: Most of them are terrestrial and predaceous; one family (Sisyridae) is aquatic, and the larvae feed in fresh-water sponges. All the larvae have thoracic legs, but no abdominal ones, well-developed heads, and mandibulate mouthparts.

KEY TO FAMILIES

1. Front legs with apical segments enlarged for grasping, fig. 286 . . . . Mantispidae
   Front legs with apical segments slender, same as other legs, fig. 284 2
2. Wings with very few veins or crossveins, fig. 285*B*. Minute insects covered with a waxy bloom and gray in appearance . . . . . Coniopterygidae
   Wings with veins and crossveins numerous, fig. 285*C–G*. Larger insects never covered with waxy bloom 3
3. Front wings with a regular, fencelike series of 12 or more crossveins (gradate veins) between R_1 and R_s, fig. 285*E* 4
   Front wings either with only 1–5 well-separated crossveins between R_1 and R_s, fig. 285*C, G*, or R_1 and stem of R_s fused, fig. 285*D* 6
4. Antennae long and slender, fig. 284, tapering to apex . . . . . Chrysopidae
   Antennae either short and clavate or knobbed at apex 5
5. Antennae short, gradually thickened towards apex . . . . . Myrmeleontidae
   Antennae long, knobbed at apex . . . . . Ascalaphidae
6. Front wings with 2 or more branches of R_s arising from fused R_1 and R_s, fig. 285*D* . . . . . Hemerobiidae

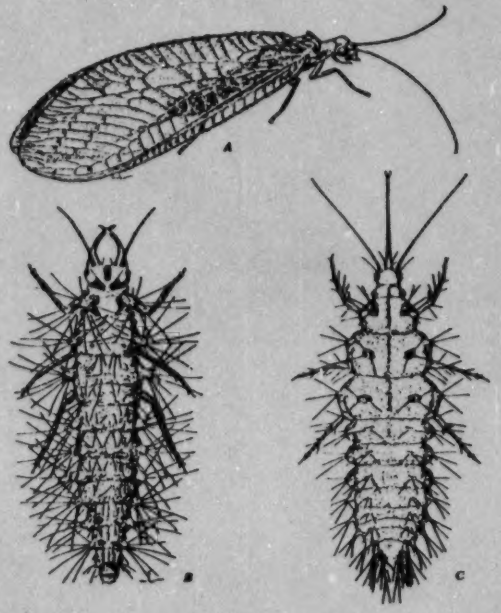


Fig. 284. Neuroptera. *A, B*, adult and larva of a lacewing (*Chrysopa* sp.); *C*, larva of a spongefly (*Sisyra* sp.) (*A, B*, from Illinois Nat. Hist. Survey; *C*, after Townsend)

Front wings with all branches of R_s arising from a separate R_s stem, fig. 285*C, F, G* 7
7. Front wings with almost all costal crossveins forked, fig. 285*F, G* 8
   Front wings with few or no costal crossveins forked and with apical margin evenly rounded, as in fig. 285*A, C* 9
8. Front wings only slightly incised and with recurrent costal vein, fig. 285*G* . . . . . Polystoechotidae

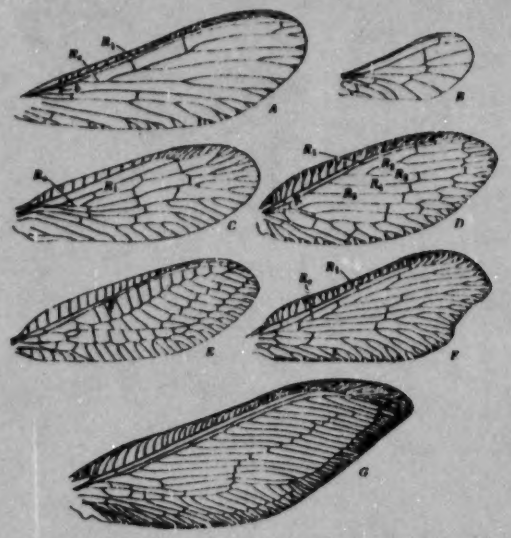


Fig. 285. Front wings of Neuroptera. *A, Nallachius*, Dilaridae; *B, Semidalis*, Coniopterygidae; *C, Climacia*, Sisyridae; *D, Hemerobius*, Hemerobiidae; *E, Chrysopa*, Chrysopidae; *F, Lomamyia*, Berothidae; *G, Polystoechotes*, Polystoechotidae. *b*, basal vein; *g*, gradate vein. (From various sources)

Front wings markedly incised and with no recurrent costal vein, fig. 285*F* . . . . . Berothidae
9. Front wings with S_c and R_1 not fused before apex; basal vein (*b*) present; fig. 285*A* . . . . . Dilaridae
   Front wings with S_c and R_1 fused some distance before apex; basal vein absent; fig. 285*C* . . . . . Sisyridae

Figure 3. Example of an Identification Algorithm in Biology

The class of outcomes to which an identification algorithm leads (see "Resultivity," p. 7) is comprised of all the subsets which make up the class--in this case, all the families of the order. This concept, of course, is termed "range" (see p. 11).

The algorithm on p. 24 is another identification algorithm. Such algorithms are much used in botany, zoology, and geology. Frequently they are called keys. Less obvious examples of this type of algorithm are those which are used to determine the rules, algorithms, or procedures applicable in a given case. Landa (1974), for example, presents identification algorithms which permit the user to determine which grammatical rule he must apply. Thus, the rule which determines the manner in which simple sentences are joined depends on the type of sentences. Figure 4 depicts an algorithm for the identification of simple sentences. Similar algorithms can be found in mathematics or in trouble-shooting manuals for mechanical or electronic equipment and in other areas.

Many of us at one time or another have been uncertain about how to punctuate a possessive noun: where do we place the apostrophe, particularly in plural nouns or in singular nouns ending in "s"? The algorithm in Figure 5 is designed to enable the user to transform nouns from the nominative case to the possessive.

We can test this algorithm by trying the following unpunctuated examples:

1. The cats fur (i.e., the fur of one cat)

2. The boys locker room (i.e., the locker room of all the boys)

3. Johnsons coat (i.e., the coat belonging to Johnson)

4. The Smiths residence (i.e., the residence of all the members of the Smith family)

The solution, in code form, for Example 1 is a b f B; for Example 2, a b f C; for Example 3, a b c d B; and for Example 4, a b c C.

This possessive form algorithm is a very simple transformation algorithm. It enables the user to transform a noun from the nominative form to the possessive form. The Euclidean algorithms in Section III, above, are examples of transformation algorithms. In theoretical terms, such algorithms enable users to change members of the domain set (or a set of inputs) into a member of the range set (or an output set).

An algorithm need not be purely identification or purely transformation. The fraction addition algorithm, shown in Figure 6 , is a mixture of the two types. Part of it (indicated by means of the heavy lines) is an identification algorithm; it enables the user to identify fractions. The remainder (indicated by means of the light lines) is a transformation algorithm; it enables the user to transform two addends into a single sum.

Domain: The five types of simple sentences (in Russian)
Range: Names of the five types: Definite personal (I and II), Elliptical, Indefinite personal, and Impersonal.
Entry skill: Distinguish between subject and predicate; conjugate verbs.

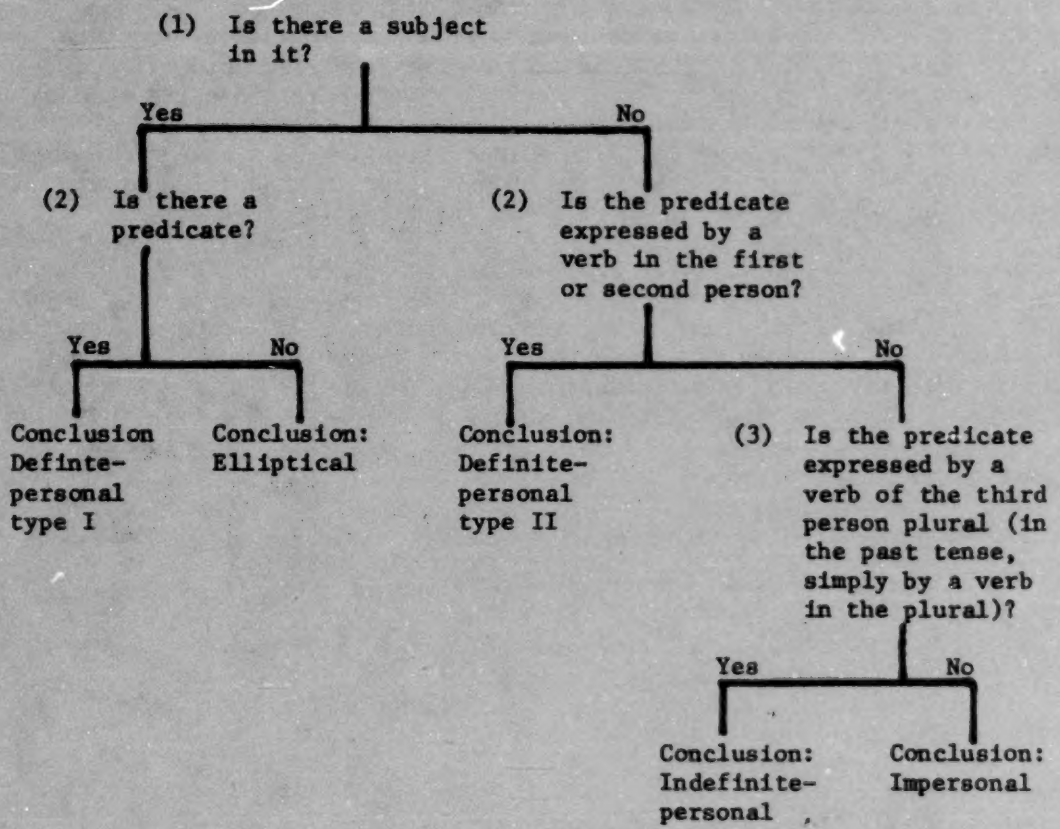

(1) Is there a subject in it?
Yes
No
(2) Is there a predicate?
(2) Is the predicate expressed by a verb in the first or second person?
Yes
No
Yes
No
Conclusion Defintepersonal type I
Conclusion: Elliptical
Conclusion: Definitepersonal type II
(3) Is the predicate expressed by a verb of the third person plural (in the past tense, simply by a verb in the plural)?
Yes
No
Conclusion: Indefinitepersonal
Conclusion: Impersonal

Figure 4. Identification Algorithm after Landa (1974), p. 437

Domain: Vocal or subvocal expressions including a possessive noun and the object of the possession.
Range: Written possessives.
Entry skill: Ability to write the nominative form of a noun after hearing or saying the possessive form.

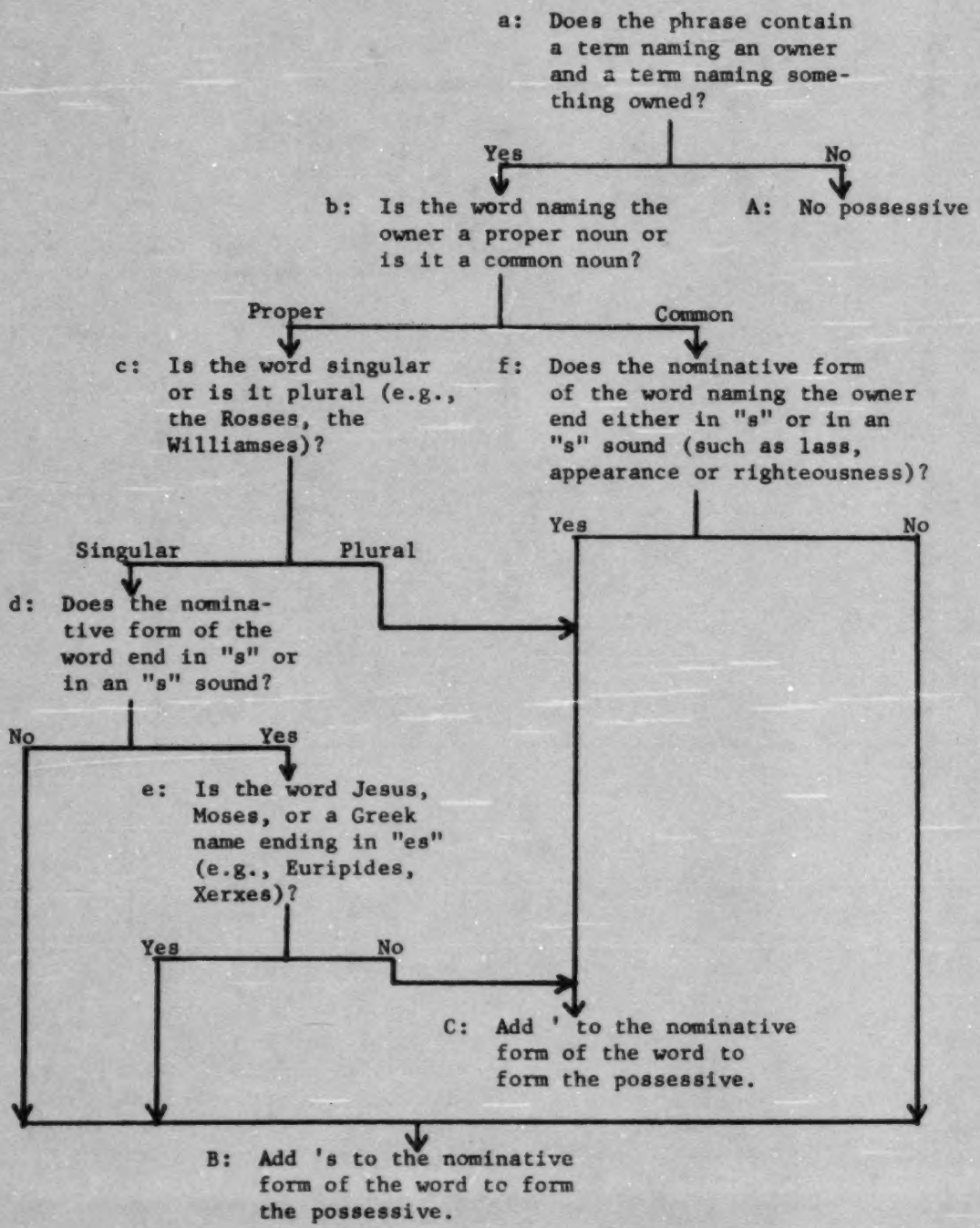

a: Does the phrase contain a term naming an owner and a term naming something owned?
Yes
No
A: No possessive
b: Is the word naming the owner a proper noun or is it a common noun?
Proper
Common
c: Is the word singular or is it plural (e.g., the Rosses, the Williamses)?
f: Does the nominative form of the word naming the owner end either in "s" or in an "s" sound (such as lass, appearance or righteousness)?
Singular
Plural
Yes
No
d: Does the nominative form of the word end in "s" or in an "s" sound?
No
Yes
e: Is the word Jesus, Moses, or a Greek name ending in "es" (e.g., Euripides, Xerxes)?
Yes
No
C: Add ' to the nominative form of the word to form the possessive.
B: Add 's to the nominative form of the word to form the possessive.

Figure 5. Algorithm for Forming the Possessive of English Nouns

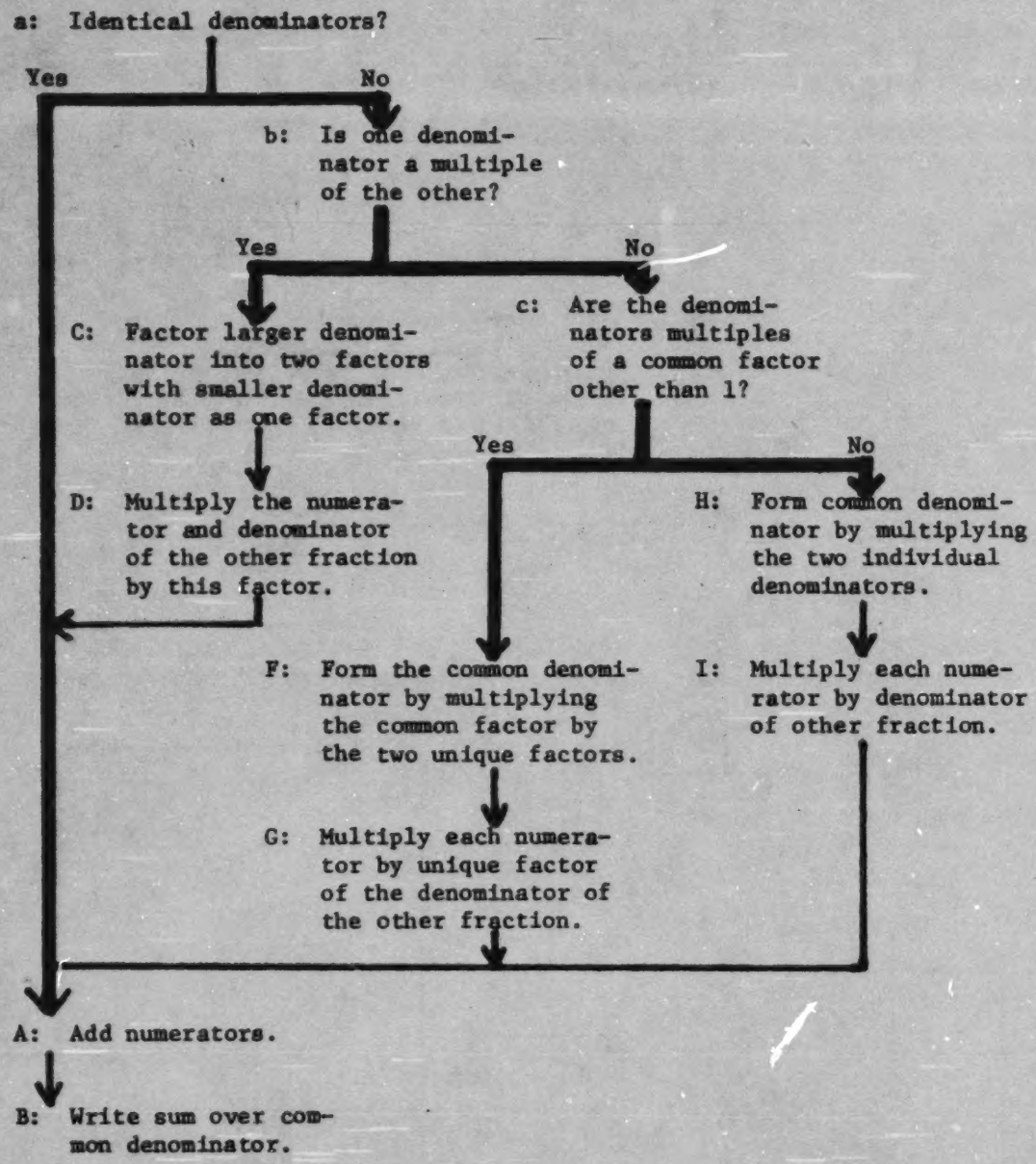
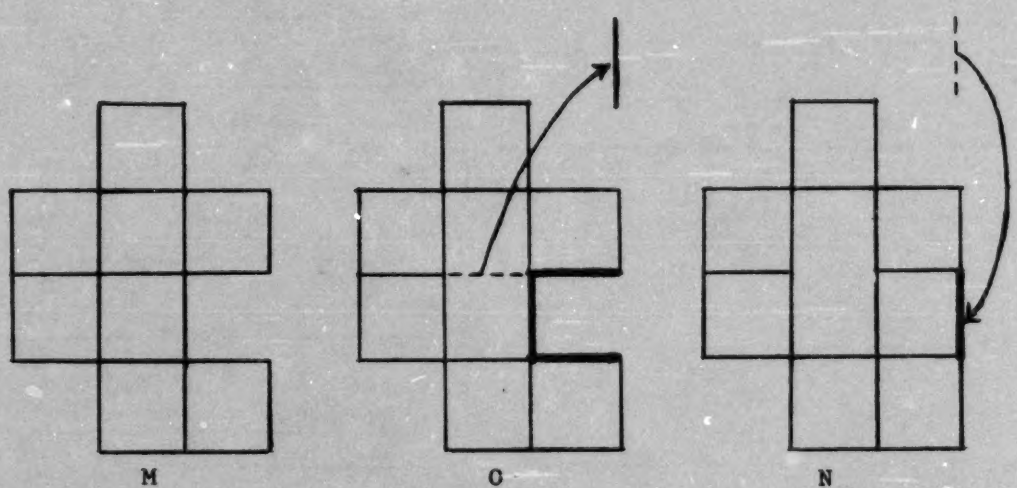

a: Identical denominators?
Yes
No
b: Is one denominator a multiple of the other?
Yes
No
C: Factor larger denominator into two factors with smaller denominator as one factor.
c: Are the denominators multiples of a common factor other than 1?
Yes
No
D: Multiply the numerator and denominator of the other fraction by this factor.
H: Form common denominator by multiplying the two individual denominators.
F: Form the common denominator by multiplying the common factor by the two unique factors.
I: Multiply each numerator by denominator of other fraction.
G: Multiply each numerator by unique factor of the denominator of the other fraction.
A: Add numerators.
B: Write sum over common denominator.

Figure 6. Algorithm for Adding Fractions

Landa (1974) refers to a third type of algorithm, one which enables the user to discover previously unencountered identification and/or transformation algorithms. He calls this a search algorithm and illustrates it by means of an example in which a learner is taught how to turn on a given machine. One method of teaching the learner this skill would be to make use of a transformation algorithm designed for the task. Landa contrasts this method with one in which the learner uses a search algorithm.

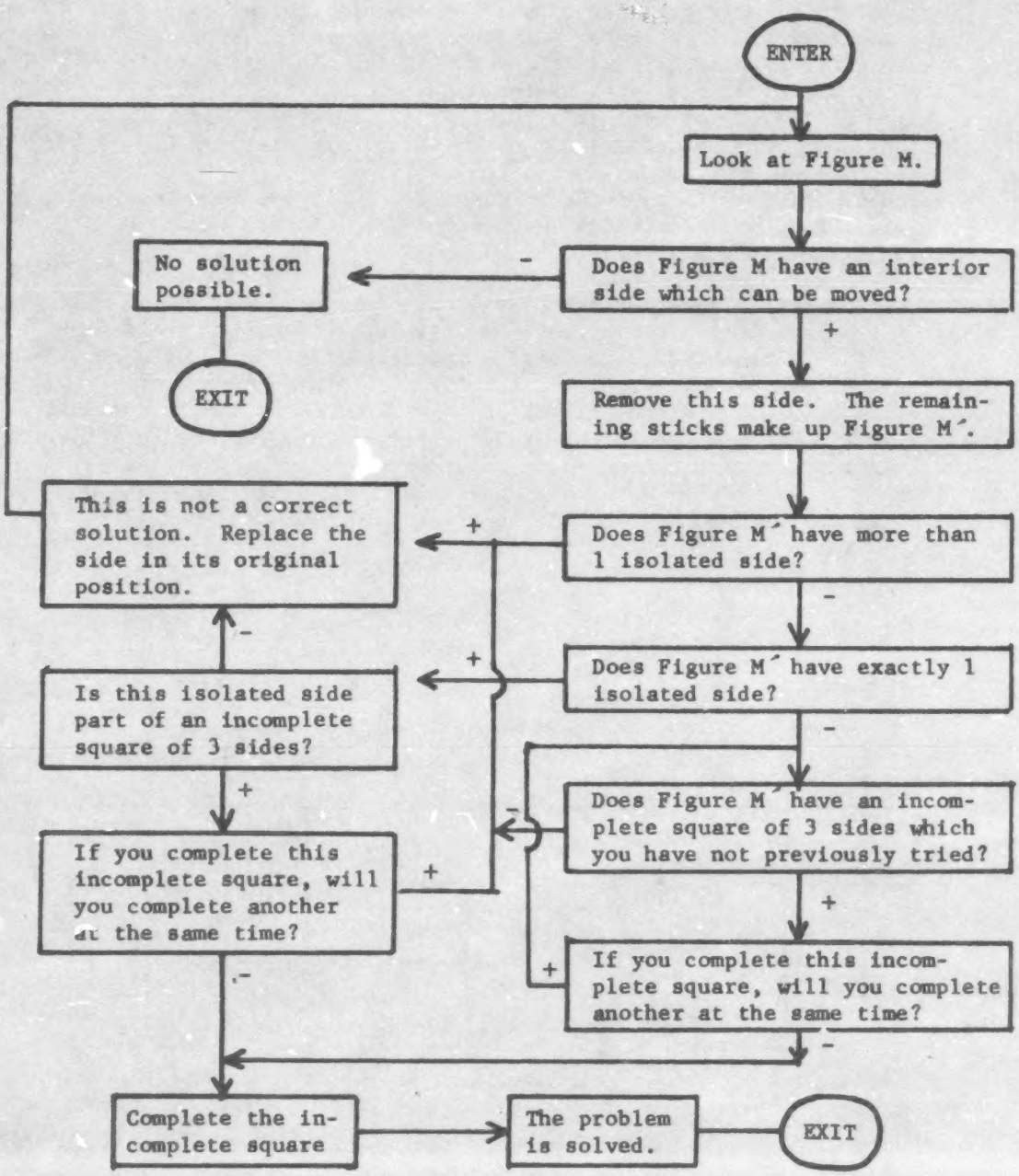
> A second method of instruction is to supply the student with some search algorithm without explaining the algorithm of solution to him, pointing out, for example, what sequence of actions he must try to perform with the apparatus in order to find the unknown rules for turning it on and for verifying that it is in working order (i.e., discover the algorithm of solution). In that algorithm, for example, there could be indications of the type: At first try pressing button a, then b; if nothing happens, then try placing the switch in the up position, then push button a, and so on. In an algorithm to search for another algorithm, all possible operations and their sequences must be foreseen. In carrying out these operations, the student necessarily discovers which sequence of operations leads to the goal, i.e., uncovers the algorithm of solution which he can then apply to any equipment of the same design. (p. 133)

Unfortunately, Landa doesn't provide a more specific example. Neither does he report any empirical investigations dealing with search algorithms. Bussmann (1971) reports an investigation of the effect of teaching learners search algorithms for the solution of a type of "puzzle" problem. The Katona (1940) match stick problem, which Bussmann used, is a classic problem in the study of learning and retention. The subject is given a figure consisting of a number of adjoining squares. His problem is to reduce the number of squares to one less than the original by moving one and only one match stick. Below is an example of the Katona match stick problem:



One of Bussmann's algorithms for the solution of Katona match stick problems is as follows:

Domain: All Katona match stick problems in which the movement of one side to a new position reduces the number of squares by one.
Range: The side to be moved; the new position of that side.
Entry skill: Identify interior side, isolated side, incomplete square.


ENTER
Look at Figure M.
Does Figure M have an interior side which can be moved?
No solution possible.
EXIT
Remove this side. The remaining sticks make up Figure M´.
Does Figure M´ have more than 1 isolated side?
Does Figure M´ have exactly 1 isolated side?
Does Figure M´ have an incomplete square of 3 sides which you have not previously tried?
If you complete this incomplete square, will you complete another at the same time?
Is this isolated side part of an incomplete square of 3 sides?
This is not a correct solution. Replace the side in its original position.
Complete the incomplete square
The problem is solved.
EXIT

Bussmann clearly ascribes the qualities of Landa's search algorithm to both his Katona algorithms.[8] However, it is clear that the algorithm presented here is one which yields solutions to a given class of Katona problems. Consequently, it cannot be a search algorithm in the Landa sense. A search algorithm does not produce a solution, per se; rather, it produces an algorithmic method for arriving at solutions. Bussmann's algorithm is clearly a transformation, not a search, algorithm in the sense in which we have defined the two terms.

Although we lack the ability to present examples of search algorithms at this time, we are convinced that further study of this kind of algorithm is an extremely promising area for future research and development. Perhaps instruction in algorithmic search methods could result in a learner's acquisition of a generalized problem-solving ability. It is even conceivable that such instruction could significantly influence mental development. Further speculation on this subject is reserved for Section VII, below.

Algorithms may also be classified as functional or control algorithms. Landa (1974) defines a functional algorithm as "an algorithm by which the operation of a system is carried out." If this system requires the intervention of a higher-order system for any reason and if this second system intervenes in a predictable, algorithmic manner, then this second system acts according to a control algorithm. The two concepts are relative. A student solving an equation in a programmed textbook follows a functional algorithm. The path he is to follow through the program is a controlling algorithm with respect to the subsystem "student" but it is a functional algorithm with respect to the higher-order system "textbook and student." Textbook and student, as seen from the standpoint of the teacher, follow a functional algorithm as long as all goes well. As soon as the student needs additional help, the teacher intervenes; he may do so algorithmically, i.e., according to an algorithm which is a control algorithm for the system "student-textbook" but a functional algorithm for the system "teacher."

The functional algorithm for the teacher is the algorithm by which the teacher functions (as long as his behavior is lawful). In more general terms, the functional algorithm of the teaching system (the programmed text, the computer, the teacher, etc.) is called a teaching algorithm (Lansky, 1969; Landa, 1974); it is in distinct contrast to the functional algorithm of the learning system, which is called a learning algorithm (Lansky, 1969). The latter term is also used for algorithms the learner is supposed to learn. In order to avoid confusion of these two fundamentally different classifications, Bung (1969) has suggested the term subject matter algorithm to designate a learning algorithm which the learner is supposed to learn.

---

[8] Although Bussmann's article appeared in 1971, he had access to the original Russian as well as the German translation of the work which we refer to as "Landa 1974."

To summarize:

A functional algorithm is an algorithm by which a user functions; one by which the operation, or function, of a system is executed.

A control algorithm is an algorithm which controls the user's path through another algorithm.

A teaching algorithm is the functional algorithm of a teaching system.

A learning algorithm is the functional algorithm of a learning system.

Landa (1974) distinguishes between an algorithmic process, an algorithmic prescription and an algorithmic description. A computer, for example, is involved in an algorithmic process when it executes a program. If this program is in a form that the computer can read (such as punched cards, for example), then the program controls the process and it is an algorithmic prescription. If the program does control the process and if it can be used for communication only, it is an algorithmic description. A human brain may or may not function as deterministically as a computer, but when humans do behave in a lawful and predictable manner, they are engaged in algorithmic processes, or at least in quasi-algorithmic processes. If one does so intentionally and consciously by following an explicit procedure, one is following an algorithmic prescription. If a person does so without conscious intent and awareness, his activity may be amenable to algorithmic description, but it does not necessarily follow an algorithmic prescription. The rules of grammar, for example, are followed correctly both by people who know them and can state them and by people who cannot do so. Both kinds of people engage in algorithmic processes, but the rules of grammar are algorithmic prescriptions for the former only, even though they are algorithmic descriptions for both.

To summarize:

When a user consciously and intentionally applies an algorithm, he is following an algorithmic prescription.

When a user, without conscious intent or awareness, applies an algorithm, he is not following an algorithmic prescription even though his activity may be amenable to algorithmic description.

VI. The Uses of Algorithms in Instruction

Algorithms as Aids to the Learner

We have already seen that an algorithm tells the user exactly what to do. An algorithm makes it possible for any user who possesses the requisite entry skills to solve correctly any problem of a given class of problems. Furthermore, this user solves the problem by using precisely the procedure, and only that procedure, which the algorithm writer intended. The error potential attributable to misinterpretation is always minimal since the representation of an effective algorithm is always simple and unambiguous. Given an adequate algorithm, someone with little or no pretraining should be able to perform correctly, adequately, and consistently, even when he is confronted with tasks as difficult as troubleshooting and repairing complex equipment, preparing and interpreting reports, or evaluating performances.

If an algorithm is represented in an appropriate form, the user is spared the waste of time which occurs when he must read both the relevant and the irrelevant contingencies. In the algorithm for forming possessives, the user is spared the trouble of reading two unneeded discriminators and one unneeded operator whenever he is dealing with a common noun. If the same algorithm were presented in discursive text form, it is highly improbable that he could avoid reading all the text. Note, however, that this efficiency is a function not of the algorithm per se, but rather of the form of representing the algorithm.

We pointed out in the section on representation that an algorithm need not be completely read before a user begins to apply it. Even more important, it need not be completely understood in order for a user to apply it. This is not to be construed as meaning that we are endeavoring to assist learners in solving problems without understanding the procedures they employ. On the contrary, an algorithm enables a user to develop understanding, little by little, as he sees a process work for him. Consequently, algorithms permit successful application and understanding to develop simultaneously.

Many algorithms are self-sufficient performance aids. Given the minimal instruction in how to "read" the representation form, such as a generalized flow chart, the user frequently needs no other assistance in mastering the skill which the algorithm is designed to implement. Most algorithms are excellent examples of "self-instruction."

Algorithms may be used to enable a learner to check the accuracy of a diagnosis or a prescription which he has made. In this context, an algorithm enables a learner to monitor his own performance effectively. Consider the key for selecting evaluation models, Figure 7, (from Horst, Talmadge, and Wood, 1975). The student has learned what the five models are and how to choose one of the five when confronted with a summary account of a project. As he acquires skill in diagnosing the nature of a project and in selecting a model, he learns to check the validity of his selection by comparing it with the result obtained when he uses the key. Obviously, the discriminators in this kind of

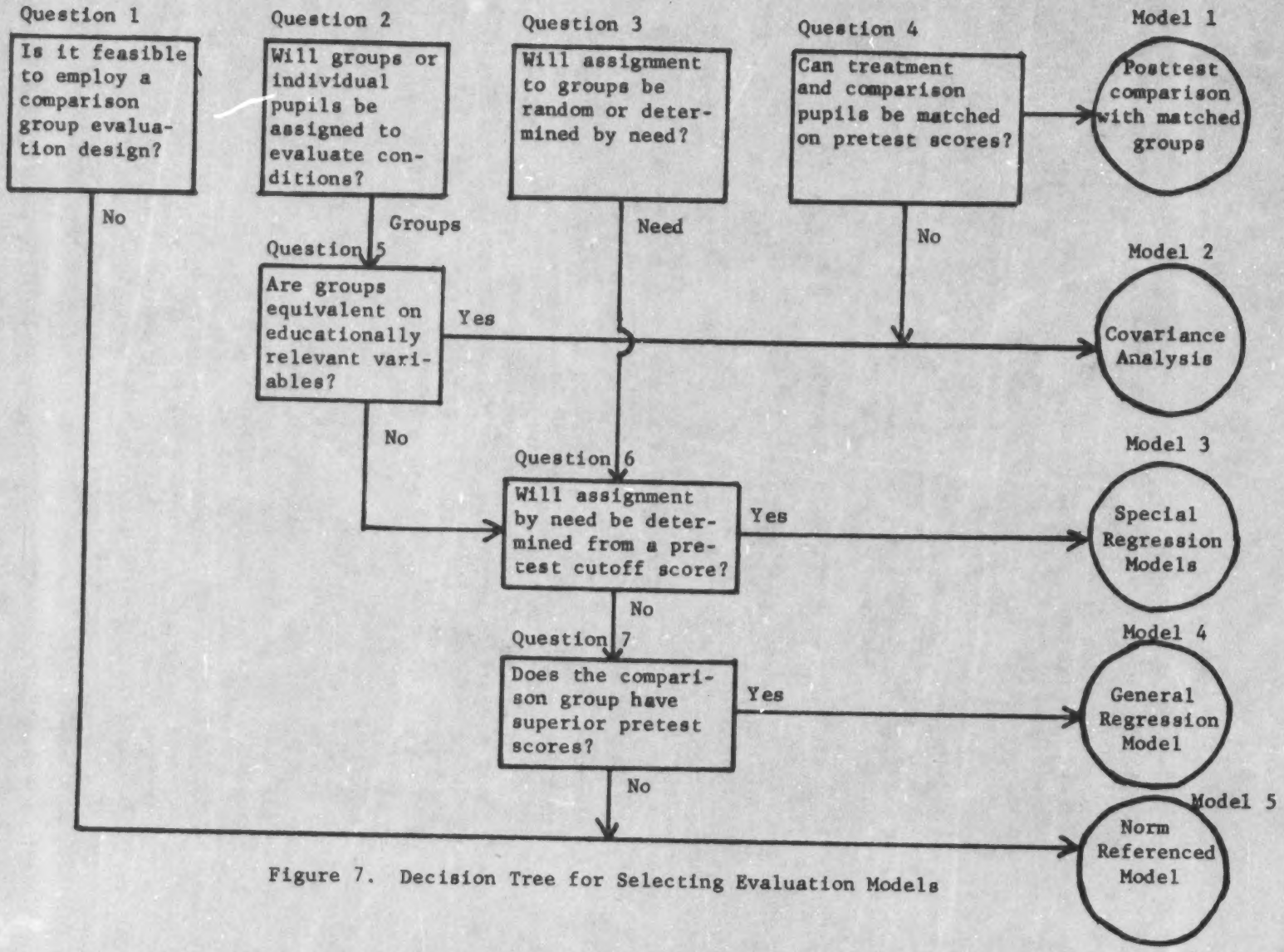


Figure 7. Decision Tree for Selecting Evaluation Models

algorithm are extremely abstract; consequently, the algorithm is used after the student has acquired the concepts in the seven boxes. This algorithm would not be appropriate in a self-instructional setting.

Algorithms as Aids to the Instructional Designer

In addition to the uses we have previously described, it appears that algorithms may also be used to aid instructional designers. We base this hypothesis on the observation that, in many instances, algorithmic procedures and instructional design procedures seem to be closely related. Some of these relationships are discussed in greater detail below.

Algorithms and objectives. Given an algorithm which has been developed with some nontechnically stated instructional goal in mind, it is very easy to derive a correct, technically stated, and easily communicated specific terminal objective from it. Let us assume that an instrucional objective always has these three elements: conditions, performance, standards. The domain of an algorithm is essentially equivalent to the conditions and the range of an algorithm is essentially equivalent to the performance. Thus, with a few minor changes and the addition of some standards, a statement of the domain and range of an algorithm can be transformed into a behavioral objective. For example, consider the algorithm for adding fractions (Figure 8). By adding the word "given" to the statement of the domain of the algorithm, we obtain the conditions portion of our outcome:

> "Given any set of two fractions with whole number denominators . . ."

A slight alteration of the statement of the range of the algorithm provides us with the performance portion of our outcome:

> ". . . the student will compute the sum of the set . . ."

By adding a statement of standards (such as "correctly"), we have a clear and concise behavioral objective:

> "Given any set of two fractions with whole number denominators, the student will compute the sum of the set correctly."

The relationship between algorithms or algorithmic formulations and objectives is, of course, based on the fact that both represent descriptions of terminal behavior. Ideally algorithms are explicit unambiguous descriptions or prescriptions, while objectives are summarized descriptions which, ideally, are also unambiguous. The interdependence of algorithms and objectives is unexplored territory which may yield interesting research problems.

Entry skills and learning hierarchies. An explicitly formulated algorithm makes it possible to "read off" required entry skills with a degree of precision and thoroughness not available with other procedures. Since many algorithms include subroutines and/or represent subroutines

Domain: Any set of two fractions with whole number denominators
Range: Sum of any set of the domain
Entry skill: Factoring of natural numbers

Example: $\frac{3}{16} + \frac{2}{3} = \frac{9 + 32}{48} = \underline{\underline{\frac{41}{48}}}$; (Path: a b c H I A B)

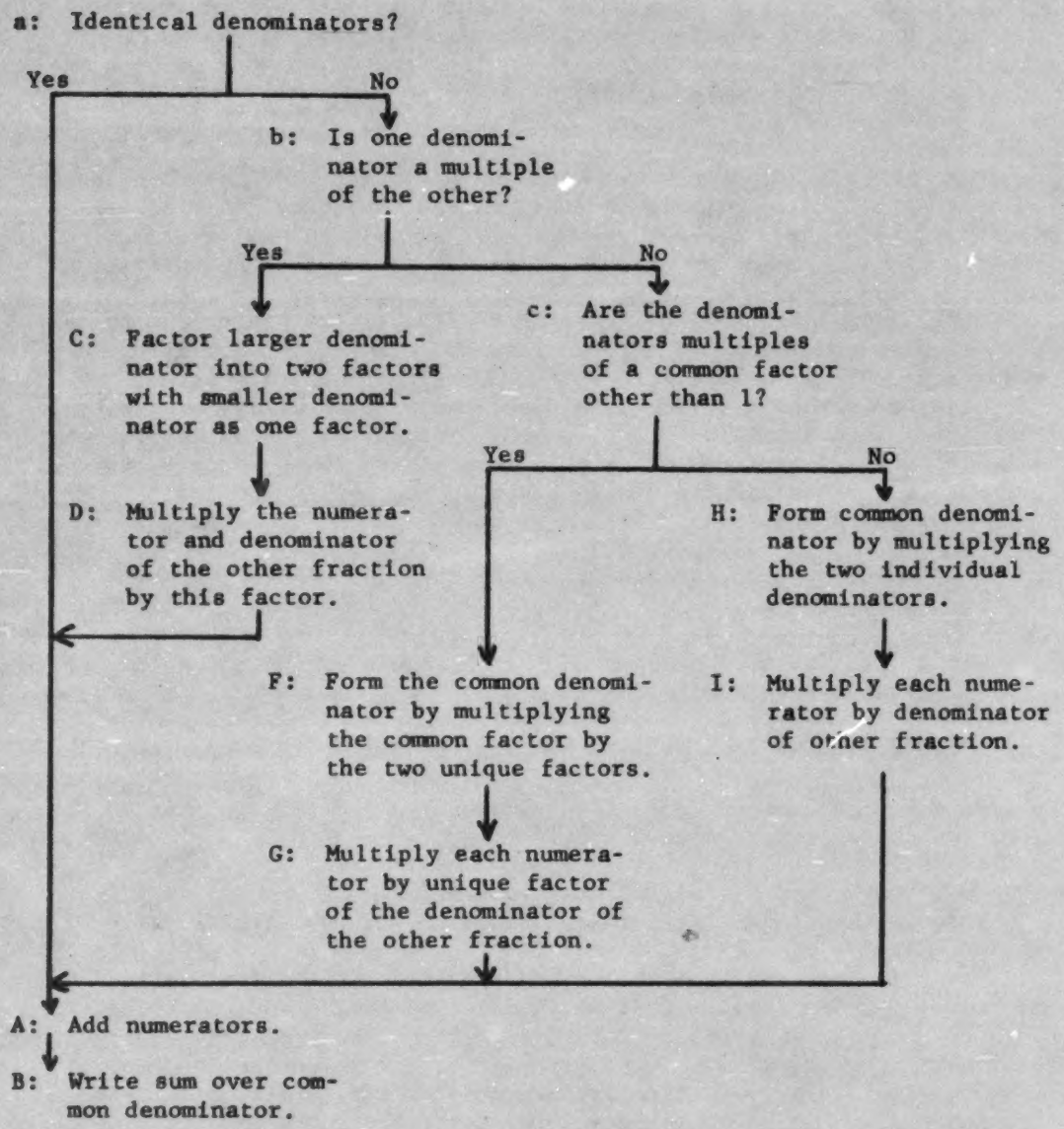

a: Identical denominators?
Yes
No
b: Is one denominator a multiple of the other?
Yes
No
C: Factor larger denominator into two factors with smaller denominator as one factor.
D: Multiply the numerator and denominator of the other fraction by this factor.
c: Are the denominators multiples of a common factor other than 1?
Yes
No
F: Form the common denominator by multiplying the common factor by the two unique factors.
G: Multiply each numerator by unique factor of the denominator of the other fraction.
H: Form common denominator by multiplying the two individual denominators.
I: Multiply each numerator by denominator of other fraction.
A: Add numerators.
B: Write sum over common denominator.

Figure 8. Algorithm for Adding Fractions (Version 1)

themselves for higher order algorithms, it is also possible to determine learning hierarchies by placing the derived entry skills (subroutines) into an order which shows the dependent and independent relationships among them. Both the determination of entry skills and the determination of a learning hierarchy are demonstrated below.

Given the algorithm for adding fractions (Figure 8), a designer can infer those concepts and skills which must be part of the learner's repertory if he is to be able to execute this particular algorithm:

Concept hierarchy:	Skill hierarchy:
fraction	factoring whole numbers
numerator	multiplying whole numbers
denominator	adding whole numbers
multiple	
factor	

The list of skills is ordered by skill levels. Factoring is clearly the highest level skill and requires skills in multiplication as a prerequisite while multiplication requires addition skills as a prerequisite. However, this ordering does not imply that the skills listed below the highest level skill represent a complete list of all prerequisites for the highest level skill.

Such a list or hierarchy can now be developed by writing an algorithm for the highest level skill (in this case "factoring whole numbers"). This algorithm again yields an ordered list of prerequisite skills, for each of which an algorithm may need to be written. Thus, we arrive at a hierarchical order of algorithms, each accompanied by an ordered list of prerequisite skills. In order to avoid the problem of an infinite regress, the designer, at some point, must decide that some particular skill must be part of the learner's repertoire. Until that particular point is identified, any skill in a "higher" list must eventually show up in a "lower" list or if it does not eventually show up as one of the subalgorithms, a branch in the hierarchy is indicated and a separate algorithmic analysis of these skills must be performed.

The final outcome is a very precise hierarchy of algorithms or of learning tasks which should include very little if any subjective developer bias. Parallel with the development of the skills hierarchy is the development of a vocabulary or concept list. This list is also ordered, inasmuch as any given new concept requires other prior concepts for its definition and explanation. The list of concepts for a given set of algorithms, therefore, represents a second hierarchical network which is assumed to be complementary to the network of skills. The two

hierarchical methods together represent a very complete basis for making decisions on sequencing the components in the ultimate instructional product.

Two things should be pointed out which bear on the applicability of the scheme outlined above. First of all, the range of applicability is limited to subject matter areas or topics within subject matter areas which are amenable to algorithmization, i.e., which have sets of interdependent rules or procedures as content. For example, most of the subject matter in history is not amenable to algorithmization because the content does not generally consist of interdependent rules or procedures. It may be possible to identify isolated bits and pieces of algorithmic subject matter in history and thus benefit in some small measure from making these algorithms explicit, but sequencing decisions on a larger scale will have to be made on the basis of criteria other than hierarchies developed by algorithmic analysis. Secondly, in the literature we find very little empirical validation and very little in-depth comparison with other similar approaches, such as the one demonstrated by Ehrenpreis and Scandura (1972), which deals with the relationships of rules and higher order rules that could be utilized in the construction of a mathematics curriculum. It is not entirely clear on what basis the authors call their approach "algorithmic," but it is clear from their data that an in-depth analysis of logical relationships exiting within a subject matter domain can lead to the elimination of a great deal of redundancy and thus to greater instructional efficiency.

Scandura (1971) identified "several hundred" rules for several hundred tasks in a mathematics curriculum. An analysis of these rules showed that many of the lower order rules could be subsumed by 12 higher order rules, thereby cutting the total number of rules by approximately 50 percent. A comparison of two groups, one of which had learned the original set of rules and the other the reduced set of rules, showed essentially that ". . . the higher order rules group was taught less but learned more." This result leads us to wonder whether algorithmic analysis may lead to similar gains in instructional economy in other fields. Algorithmic analysis is somewhat broader and more inclusive than Scandura's approach, since it includes the parallel development of a concept hierarchy. It may, therefore, be an even more effective overall approach--a hypothesis that should be put to the empirical test.

Prompting. Prompting is a technique which instructional designers use frequently; so is the gradual withdrawal, or fading, of prompts. Algorithms are highly amenable to the gradual withdrawal of a prompt. Take for example the fraction adding algorithm (See page 42): after the pupil has added several pairs of addends correctly, one or another of the discriminators or operators can be covered or removed and the pupil continues summing pairs of fractions. This process is repeated until the pupil can find the sum of a pair of fractions without referring to the flow chart or any part of it. Remember: it is the semantic, not the syntactic, elements of the algorithm which are gradually withdrawn or faded. The complete algorithm, we trust, becomes a part of the pupil's cognitive structure; stored in his long-term memory, it can be used whenever the need arises.

Figure 9 provides an illustration of the technique. The problem is to construct a magic square--one in which the sums of all columns, all rows, and all diagonals are identical; thus:

4	3	8	= 15
9	5	1	= 15
2	7	6	= 15
15	=15	=15	=15 15

Try the algorithm, starting with any whole number except 3. Then try the abbreviated version, Figure 10, from which many of the prompts have been withdrawn. Next, try Figure 11, which is even more "faded" than the previous version. Finally, it is entirely conceivable that you can now construct a magic square without reference to any of the three figures. (Again--remember that the representation of the algorithm has been faded, not the algorithm itself. You probably still use the algorithm, which has now been stored in your memory, to solve the problem.)

Individualized instruction. Since nearly every task can be accomplished in more than one way, it is usually possible to develop more than one algorithm for one and the same task. Two algorithms which are applicable to the same kinds of problems and yield the same results are called equivalent. Equivalent algorithms have the same range and domain, but different operators and discriminators and, therefore, generally require different entry skills. One and the same behavioral objective may thus be taught using one of several equivalent algorithms, depending on a student's entry skills. The Euclidean algorithm for finding the greatest common divisor of any two whole numbers is a good case in point. Version 1 (Figure 12) requires the ability to factor whole numbers, Version 2 (Figure 13) requires subtraction of whole numbers, and Version 3 (Figure 14) requires the division of whole numbers as prerequisites.

It follows that the same objective may be reached by at least three different instructional routes. This provides opportunities for individualization that go far beyond flexibility in time allotments and remedial adjustments. For example, much of a mathematics curriculum could be taught by means of a number of entirely different paths, each of which would lead to the same set of terminal objectives.

It should be noted that differences in learner traits as well as differences in specific entry skills may be accommodated by an appropriate choice of the algorithm to be taught. A learner may, for example, possess all of the entry skills required for any one of the three versions of the Euclidean Algorithm. In this instance the choice could be made on the basis of such learner traits as IQ or learning style. In any case, algorithmic analysis can open up a wider range of true choices for the optimal adaptation of instruction to the learner.

Domain: Nine different whole numbers ≥ 1; a square of nine empty cells.

Range: A 3x3 array of numbers whose column sums, row sums, and diagonal sums are identical.

Entry skill: Ability to follow written directions involving left, right, up, and down; ability to count by ones in whole numbers.*

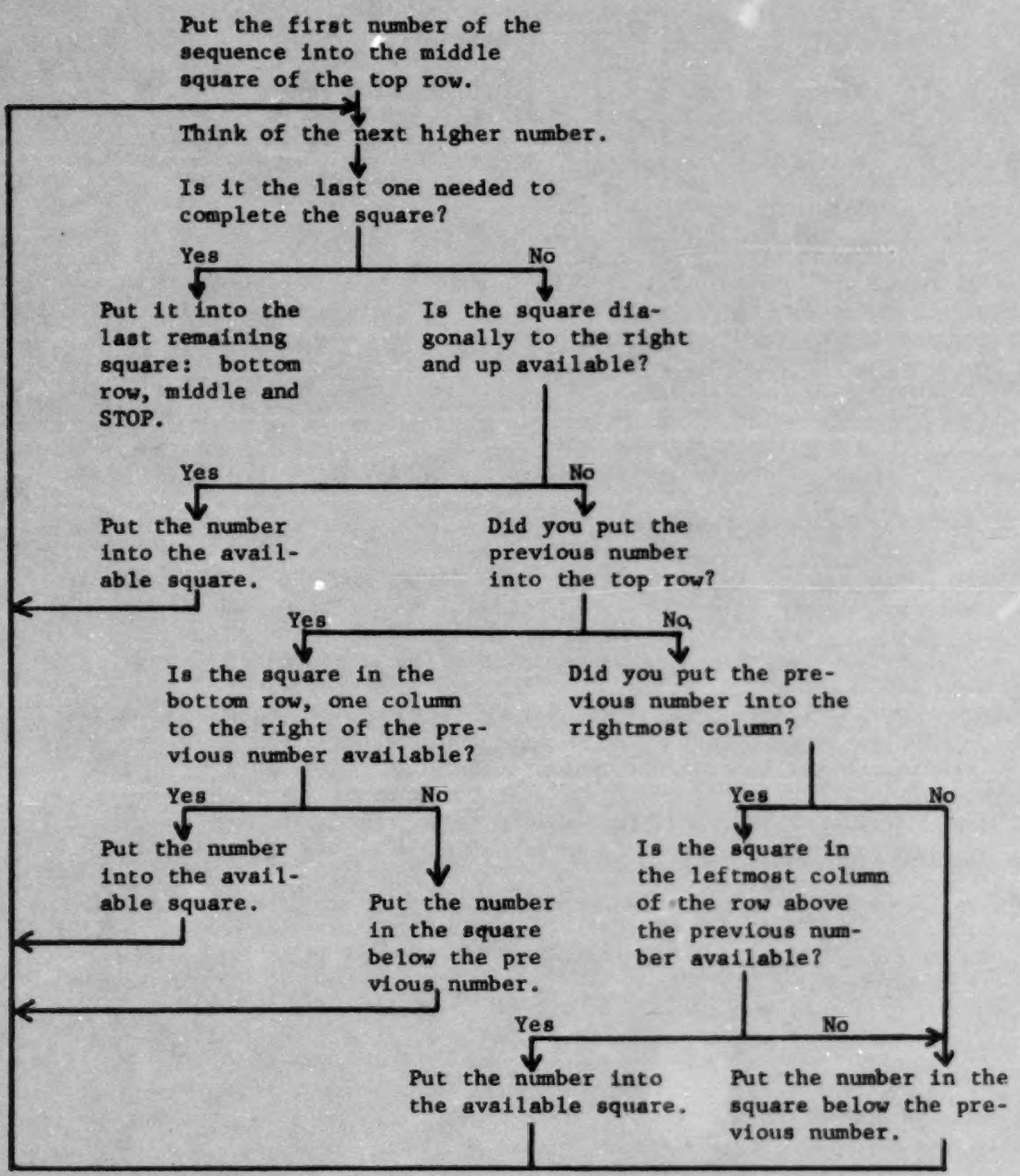

Put the first number of the sequence into the middle square of the top row.
Think of the next higher number.
Is it the last one needed to complete the square?
Yes
No
Put it into the last remaining square: bottom row, middle and STOP.
Is the square diagonally to the right and up available?
Yes
No
Put the number into the available square.
Did you put the previous number into the top row?
Yes
No
Is the square in the bottom row, one column to the right of the previous number available?
Did you put the previous number into the rightmost column?
Yes
No
Yes
No
Put the number into the available square.
Put the number in the square below the previous number.
Is the square in the leftmost column of the row above the previous number available?
Yes
No
Put the number into the available square.
Put the number in the square below the previous number.

Figure 9. Algorithm for Producing a 3x3 Magic Square

*It is interesting to note that this algorithm does not require the user to add or subtract.

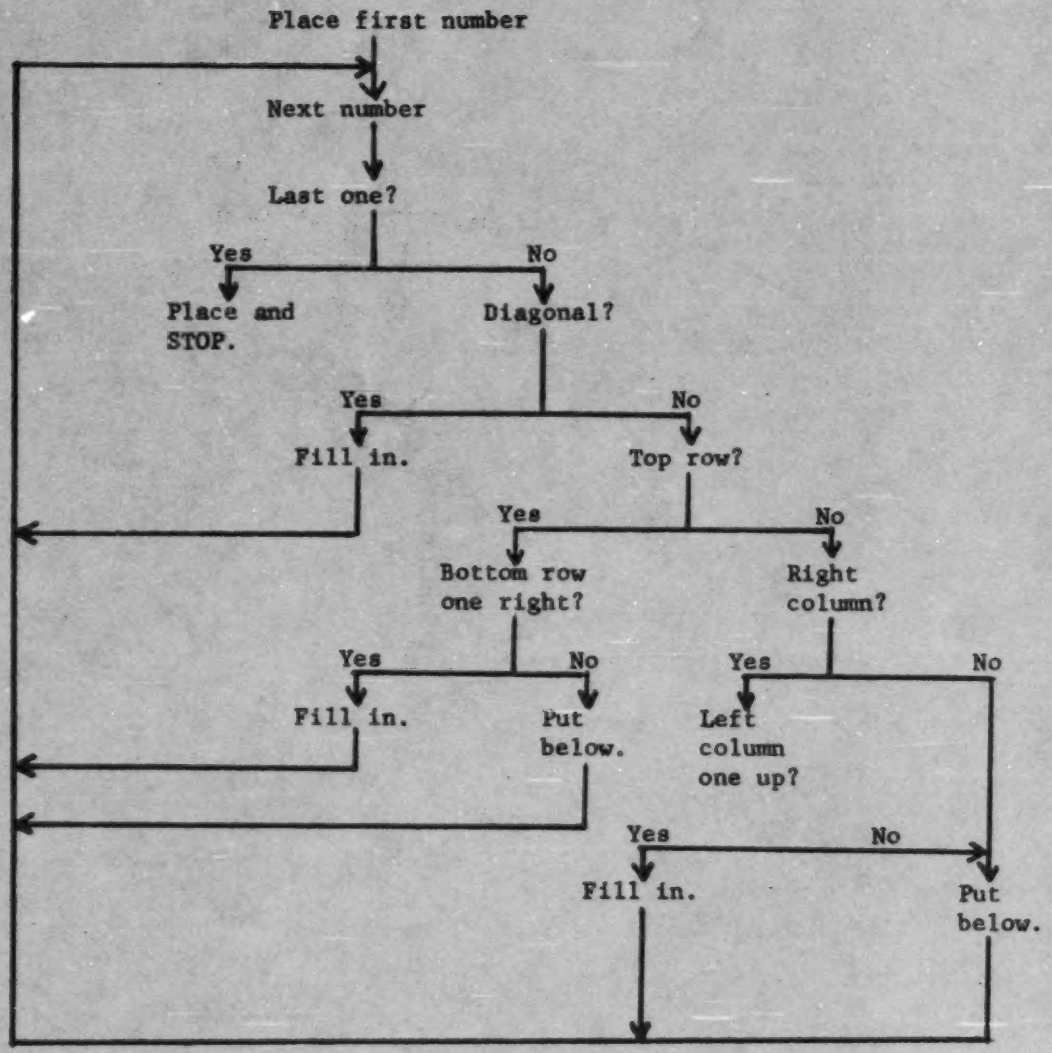


Figure 10. Abbreviated Flow Chart

(Magic Square)

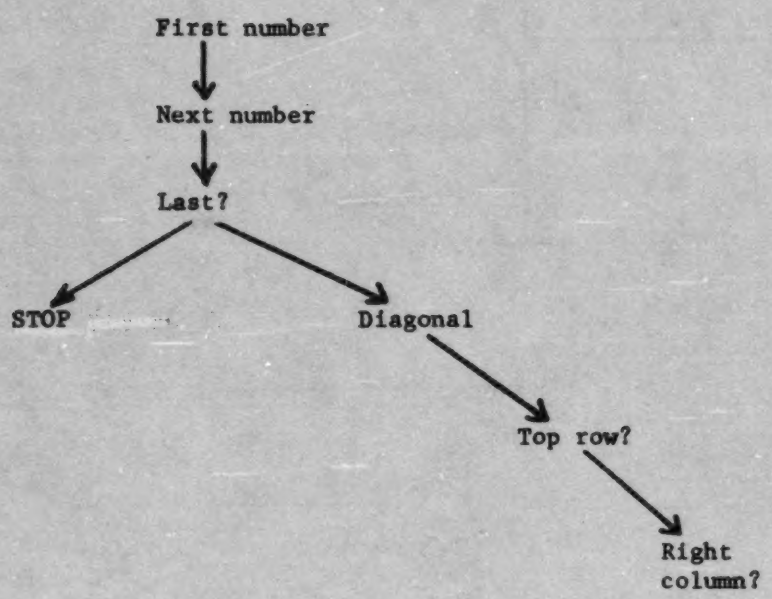


Figure 11. A Reminder

(Magic Square)

Domain: Any set of two natural numbers
Range: The greatest common divisor for any set of the domain
Entry skill: Factor natural numbers

Example: Find GCD for 144 and 32.

$$144 = 1 \times 2 \times 2 \times 2 \times 2 \times 3 \times 3$$
$$32 = \underline{1 \times 2 \times 2 \times 2 \times 2 \times 2}$$
$$1 \times 2 \times 2 \times 2 \times 2 = \underline{\underline{16}}$$

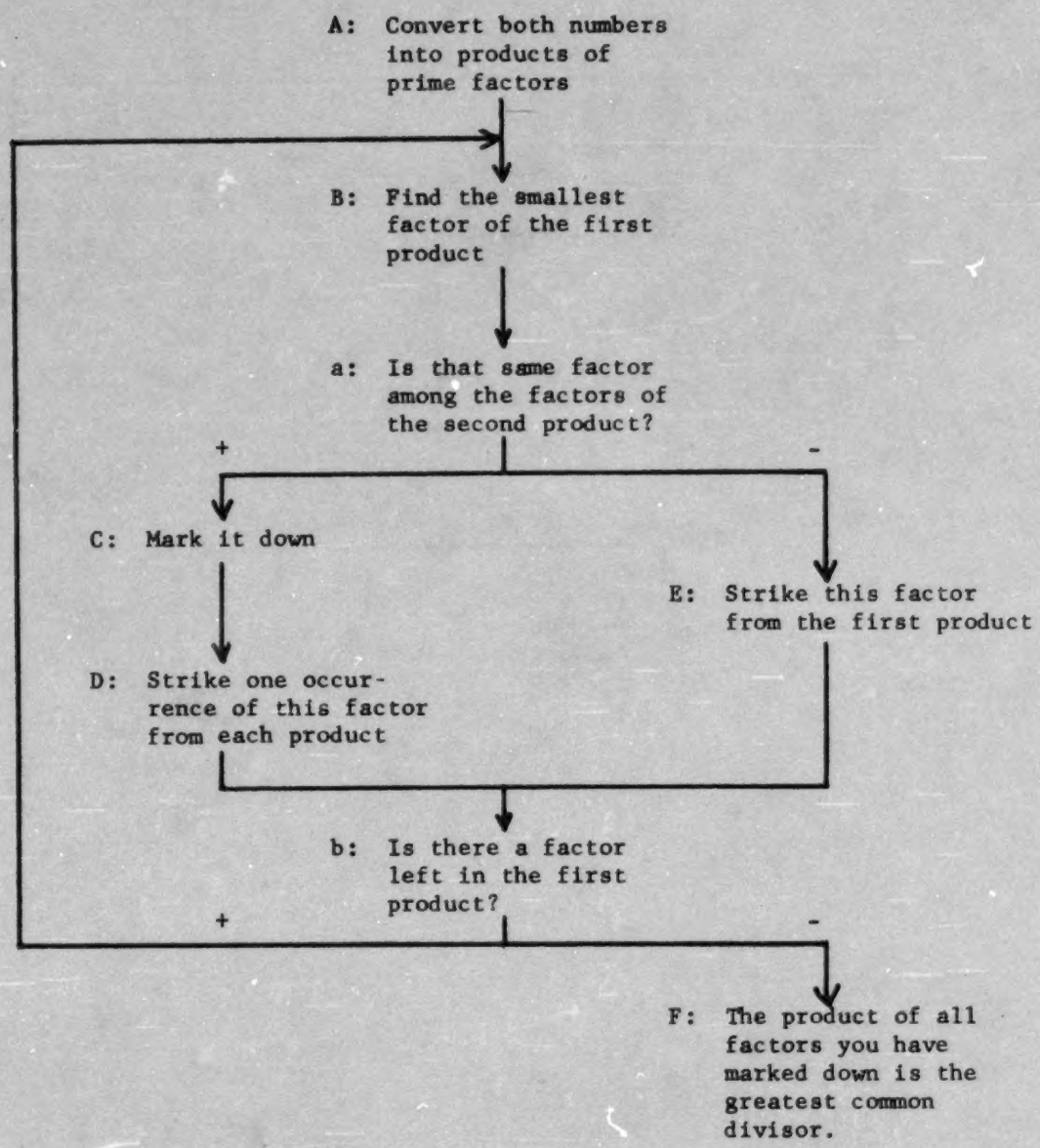

A: Convert both numbers into products of prime factors
B: Find the smallest factor of the first product
a: Is that same factor among the factors of the second product?
+
-
C: Mark it down
E: Strike this factor from the first product
D: Strike one occurrence of this factor from each product
b: Is there a factor left in the first product?
+
-
F: The product of all factors you have marked down is the greatest common divisor.

Figure 12. Euclidean Algorithm (Version 1)

Domain: Any set of two natural numbers
Range: The greatest common divisor for any set of the domain
Entry skill: Subtract natural numbers

Example: Find GCD for 144 and 32: 144 - 32 = 112
112 - 32 = 80
80 - 32 = 48
48 - 32 = 16
32 - 16 = 16

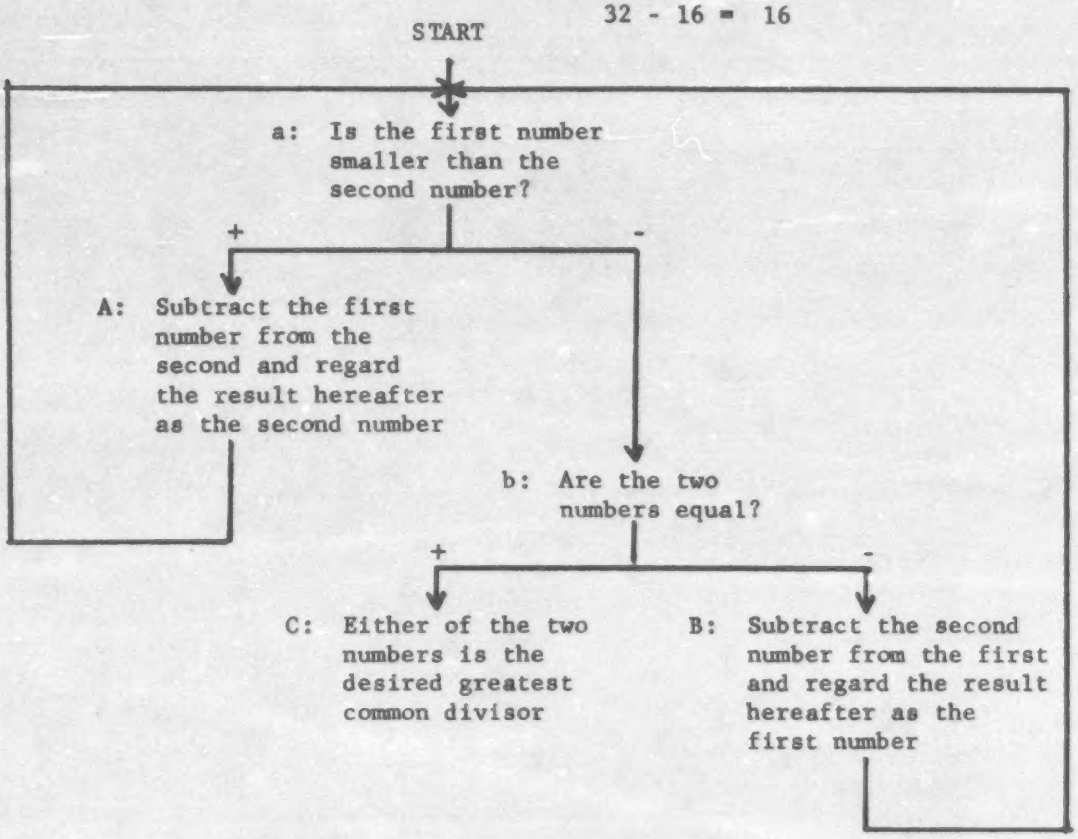

START
a: Is the first number smaller than the second number?
+
-
A: Subtract the first number from the second and regard the result hereafter as the second number
b: Are the two numbers equal?
+
-
C: Either of the two numbers is the desired greatest common divisor
B: Subtract the second number from the first and regard the result hereafter as the first number

Figure 13. Euclidean Algorithm (Version 2)

Domain: Any set of two natural numbers
Range: The greatest common divisor for any set of the domain
Entry skill: Divide whole numbers

Example: Find GCD for 144 and 32:

$$\begin{array}{r} 4 \\ 32\overline{)\,144} \\ \underline{128} \\ 16 \end{array} \quad \text{remainder}$$

$$\begin{array}{r} 2 \\ 16\overline{)\,32} \\ \underline{32} \\ 0 \end{array} \quad \text{remainder}$$

GCD = 16

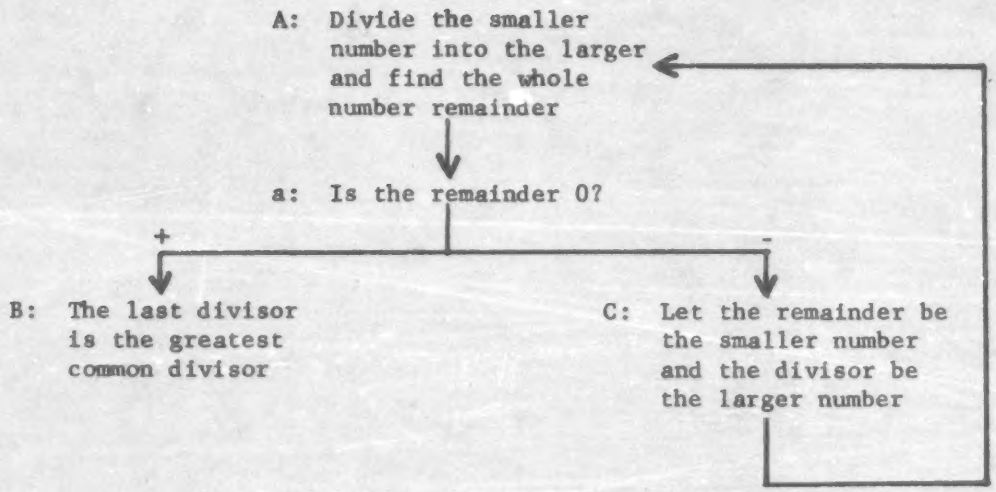

A: Divide the smaller number into the larger and find the whole number remainder
a: Is the remainder 0?
+
-
B: The last divisor is the greatest common divisor
C: Let the remainder be the smaller number and the divisor be the larger number

Figure 14. Euclidean Algorithm (Version 3)

VII. Research and Development Problems

Nearly every suggestion for the use of algorithms discussed in the preceding sections is based on logical or philosophical considerations. There is little basis for any empirical pronouncements because the literature is almost completely devoid of reports of experiments in which algorithms are an independent variable. In this section, a number of questions are raised which provide a point of departure for the formulation of researchable hypotheses in which algorithms are the independent variable. First, general questions are discussed; then questions which are particularly relevant to flying training are considered.

Algorithms for Learning and Teaching

It is obvious that some learning tasks and some teaching procedures are much more amenable to algorithmization than others. For example, one can safely assert, without empirical evidence, that it is a fairly straightforward matter to construct an algorithm for the identification of families of the order neuroptera, but that it is fairly difficult to construct one which can be used to identify certain personality traits on the basis of subjects' handwriting samples. The important question at this time is not so much one of ascertaining for which learning or teaching tasks an algorithm can be constructed; this can be determined rather simply by attempting to construct an algorithm or a set of algorithms. What is of concern at this time is whether or not a given set of algorithms facilitates learning or teaching. If algorithms are facilitative, is it possible to begin defining classes of problems for whose solution algorithms are an efficient and/or effective aid?

Perhaps the most difficult, but also a potentially extremely rewarding effort lies in the area of search algorithms. As we mentioned in Section V, a search algorithm is essentially an algorithm which enables the user to discover or formulate additional algorithms. Theoretically, it should be possible to construct search algorithms. The literature, however, provides no examples of research endeavors along this line. What are the precise characteristics of search algorithms? How are they generated? What applicability do they have? These and similar questions are among the most significant which we have been able to identify in our research thus far.

The following is a representative, not exhaustive, list of additional questions to which research efforts might be addressed:

1. Can an algorithm increase a learner's ability to generalize? If so, can the ability to generalize be facilitated by the use of algorithms when the focus of the learning task is the generalization of the structure? when the focus of the learning task is the generalization of the substantive elements?

2. Does a learner who discovers an algorithm perform better than a learner who is given an algorithm?

3. What is the applicability of algorithms to the acquisition of various types of rule-governed behavior (see Eubanks, 1976)?

4. Will differences in the representational form of a given algorithm produce differential effects?

5. What steps does a subject matter expert follow in constructing an algorithm?

Algorithms in Flying Training

There are many academic areas in the Undergraduate Pilot Training Curriculum. Examples of these academic areas include such topics as aerospace physiology, aircraft accident prevention, instruments, navigation, applied aerodynamics, and many others. Research addressed to the role of algorithms in learning and teaching such academic topics is covered in the immediately preceding paragraphs. On the other hand, flying training (particularly such areas as procedures, instruments, and navigation, in both simulators and aircraft) presents a set of problems quite different from any discussed heretofore.

One of the reasons why flying training presents a different set of problems is related to how one develops an algorithm for learning a response which cannot be unambiguously described by any verbal means. For example, we have frequently referred to the set of seven verbal cues which aid a student pilot to master the Vertical S-A. If we wished to algorithmize these cues, we would immediately be confronted with the problem of representing the first operator, "apply power at a smooth, slow, and steady rate." The verbal representation, either textual or oral, appears to be insufficient to control the learner's behavior within very precise criterion limits (Brecke, Gerlach, and Shipley, 1974). Thus, it seems unlikely that an algorithm can be constructed which would depend, wholly or partly, on such a verbal representation because resultivity would be lacking. The verbal cue permits too much response variability to insure an acceptable execution of the maneuver. Obviously research is needed which may yield a solution to this kind of problem.

Quasi-algorithmic prescriptions and quasi-algorithms. Landa (1966) uses the term quasi-algorithmic prescriptions to designate procedures which are not algorithms in the strict mathematical sense. He distinguishes quasi-algorithmic prescriptions from algorithms proper in this manner:

(1) The criteria of replicability, generality and resultivity are only approximately fulfilled by quasi-algorithmic prescriptions.

(2) It is generally not possible to unambiguously delimit the domain for quasi-algorithmic prescriptions.

(3) It may not be possible to specify a finite number of operations for quasi-algorithmic prescriptions.

It appears that it would be difficult to identify quasi-algorithmic prescriptions on the basis of the distinguishing characteristics described above. Indeed, Landa himself emphasizes that the notion of a quasi-algorithmic prescription is less precise than the notion of an algorithm. However, Landa does discusss a characteristic of algorithms which, although he apparently did not intend it to be used for such purposes, can help in the identification of quasi-algorithmic prescriptions. Landa states that an algorithm is sufficiently elementary if, and only if, the discriminators and operators are unambiguous. Operationally, this translates into the statement that a given user (or class of users) must be able to make the discriminations and perform the operations specified in the algorithm. This leads to the logical conclusion that the elementarity of an algorithm is dependent on the user. If the user is a machine, the problem of determining whether or not the algorithm is sufficiently elementary is relatively simple; indeed, there should be no reason why this elementarity, or lack of it, cannot be specified a priori. In the areas of education and psychology, the problem is quite different. Human users are less predictable than machines. Indeed, the behavior of humans is so unpredictable that frequently the elementarity of an algorithm must be determined pragmatically; it cannot be determined a priori. Add to this the fact that Landa is concerned not only with the upper limits of elementarity, but also with the lower (i.e., whether or not the algorithm is too elementary), and the problem becomes even more complex. Thus, Landa's concept of sufficient elementarity seems to lead to such frustrating complexity that one might be led to conclude that algorithms have little or no applicability to flying training.

Bung (1971), however, suggests a solution which deserves careful attention. He has attempted to deal with the subjectivity problem which elementarity poses by introducing the concept of quasi-algorithm:

> Quasi-algorithms are procedures which are explicit for, and can be carried out by, a specified set of human beings; algorithms are procedures which are explicit for, and can be carried out by automata. Since all procedures which can be carried out by automata can also be carried out by human beings but not vice versa, it follows that all algorithms are quasi-algorithms but not all quasi-algorithms are algorithms. The set of all algorithms is therefore a subset of the set of all quasi-algorithms. (p. 3)

This definition permits the inclusion of machine algorithms in the class of quasi-algorithms. From an instructional design and development point of view, nothing is gained from Bung's distinction. However, if we modify Bung's scheme for classifying algorithms and quasi-algorithms, we arrive at a very practical distinction. Let us point out that Bung's discussion of a user's ability to carry out an algorithm or quasi-algorithm is similar to Landa's discussion of the sufficient elementarity of algorithms. Keeping this fact in mind, let us divide all algorithms into two classes, those for which the attribute sufficient elementarity can be specified a priori and those for which it cannot. The former are "true" algorithms; the latter we shall call "quasi-algorithms."

Thus, it is a simple matter of logic to extend the concepts of Landa and Bung to a precise and practical (i.e., applicable to instructional design) definition: a quasi-algorithm is one for which the attribute sufficient elementarity must be determined pragmatically. This extension of the theory of algorithms is of particular significance to such areas as flying training. Many of the responses which a student pilot must learn are continuous control actions. It seems highly unlikely, given the present state of the art, that true algorithms for any of the learning or instructional problems can be formulated; at least, it does not seem practical to attempt to do so at this time. However, quasi-algorithms may provide an effective means of surmounting the difficulties imposed by the restrictive nature of true algorithms. The application of quasi-algorithms to such tasks as learning an instrument maneuver is a legitimate research and development effort.

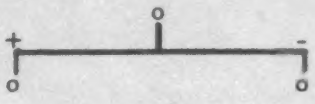
Variables in algorithms. The question of whether or not an algorithm is sufficiently elementary can be investigated on a syntactic or a semantic or a pragmatic level.

(1) Syntactic variables. We have said earlier that the macrostructure of an algorithm represents its syntax. For example, the syntactical aspects of the Euclidean algorithm (page 49) are shown in Figure 15. Another representation of the syntax of this algorithm is the following set of symbols:

A B a 2 C D b 3 B 3 F. 2 E b 4 B 4 F.

These symbols have no specific intrinsic or endemic meaning. They are not fixed to any particular subject matter (or algorithm) and could represent such diverse referents as the syntactical aspects of a procedure for operating a machine or for ascertaining whether or not a specific rule of grammar is applicable. It appears perfectly reasonable to categorize algorithms according to various syntactical features. Linear algorithms without discriminators are quite different from branching algorithms with operators only at the exit points; both differ from an algorithm with recursive loops or from an algorithm without recursive loops.

The syntax of algorithms provides a mechanism or a procedure for the quantification of certain features, such as the number of operators, discriminators, exit points, and recursive loops. These variables, as well as the structural variations mentioned above, are certainly important determinants of general and individual learners' behavior potential with respect to replicable procedure. It is quite logical to assume that syntactical complexity of an algorithm and ability factors such as IQ are directly related. An algorithm with the elementary structure:



may well be "elementary" for pupils in grade one or higher. However, an algorithm with a more complicated structure (for example, the Euclidean

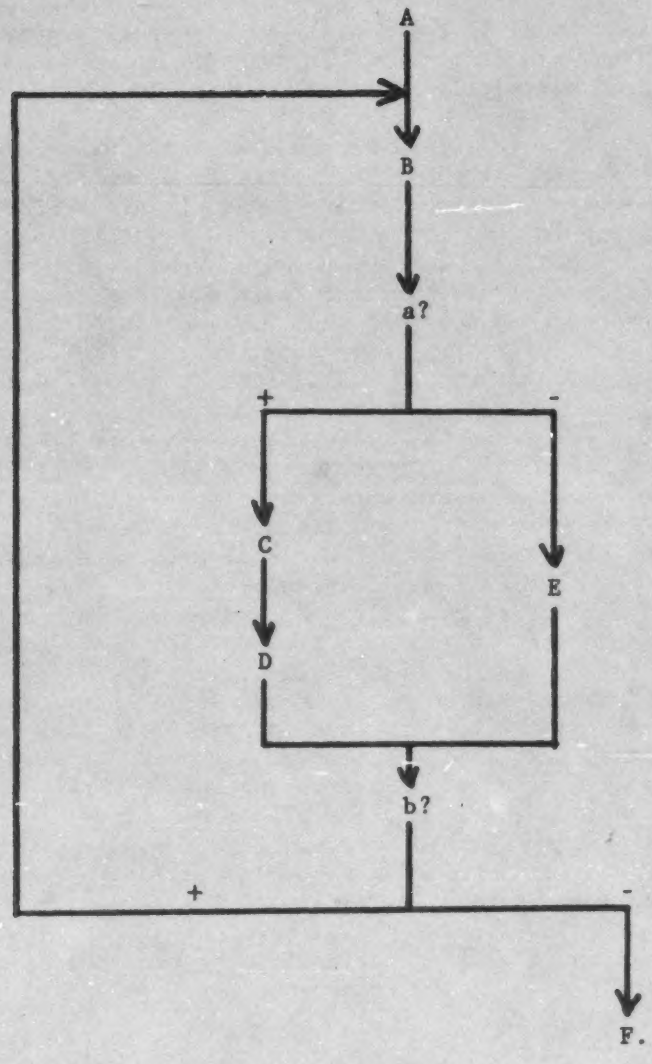


Figure 15. The Syntactical Structure of the Euclidean Algorithm (Version 1)

Algorithm) may not be elementary for the same pupils. Version 1 of the Euclidean Algorithm contains six operators, two discriminators and one recursive loop. The Shipley version of the Euclidean Algorithm (Figure 14 has three operators, one discriminator and one recursive loop. Syntactically speaking, the Shipley version is much simpler than the original, and may be more suited for students of lower ability. Syntactical variables may, therefore, be among the critical factors in determining when learners can be introduced to any particular replicable procedure.

(2) Semantic variables. When we speak of the semantic aspects of an algorithm, we refer to the meaning of the verbal elements associated with the symbols of the syntactical skeleton. Weaver (1949), in his preface to Shannon's basic treatise on information theory, expressed it thus: "How precisely do the transmitted symbols convey the desired meaning?" (p. 24). Landa refers to this aspect as content (as opposed to form). The syntactic and the semantic components of an algorithm may be separated for the purpose of illustration as well as for practical considerations. Bung's Bird algorithm (see p. 24) illustrates this division.

To ask whether or not a procedure is replicable in semantic terms means basically: Can a specified user understand what he is to do? Will every user of the specified class of users understand the meaning of each element? Will every user interpret any given element in the same way? To put it another way: Are the formal characteristics of these semantic units elementary for an identifiable class of users? Landa (1974) answers this question affirmatively and gives the following example: The truth value of the sentence, "He got off the train in Tashkent and went immediately to his Moscow apartment,"[9] can be recognized as false by anyone who has a normal command of language and the geography of the Soviet Union. It involves the basic semantic axiom that no object can be in two places at the same time. The example is, of course, designed to prove a point, but it may not be possible to reduce every sentence in a procedure to an axiomatic kernel of truth. Therefore, it will be necessary to design other means of analyzing the semantic content of a procedure. One way to accomplish this purpose would be to decompose sentences into component concepts or into the component skills implied. The operator "A" in the Euclidean Algorithm (p. 49), for example, is crucial since it could be assumed that any user for which this operator would be elementary would surely have no difficulties with the remaining operators or discriminators.

(3) Pragmatic variables. Given an algorithm which is semantically adequate we are still faced with the problem of determining a "sufficiently elementary" operation from a pragmatic standpoint. Can the user execute the operations and discriminations specified? Can he do what he is supposed to do? Can all users do it equally well, or with the same speed, force, precision? If a procedure specifies, "Advance throttle smoothly," how wide is the range of possible behaviors resulting from

---

[9]Roughly equivalent to getting off a train in Chicago and going immediately to one's San Francisco apartment.

this cue? For apprentice surgeons, the removal of an appendix may very well be an algorithmic procedure in all but the pragmatic aspects of the operation. Not every apprentice may have the combination of sensitivity and dexterity required to cut through the abdominal wall. It may be completely clear to a user what it means to factor a number; he may understand perfectly what he is supposed to do, but he may not be able to do it correctly. A person or a machine may not be able to make the required discriminations, as for example, a color-blind person who is directed to react differently to different colors. A musical score for the tuba may be algorithmic for a grown man, but a seven-year old boy may be physically unable to handle the instrument, even though he knows exactly what he is supposed to do. The pragmatic aspect, then, requires that an algorithm be operationally definable; this means that it must be demonstrable that a specified class of users can execute the algorithm in such a manner that an accetpable outcome results.

Whether or not the parts of a procedure are sufficiently elementary requires an examination of three clearly identifiable aspects. The criterion of sufficient elementariness is satisfied for a specified class of users if these users possess the prerequisite skills to unequivocally perform the syntactic, semantic, and pragmatic discriminations and operations that are called for by the element or set of elements. If the users fall short in their skills (i.e., if there is a deficit of user skills with respect to requirements of the procedure), then the distinctions made above provide the taxonomical and conceptual tools for pinpointing the exact nature of the deficit. The potential of any measure for remedying the deficit by either changing the procedure or upgrading user skills is, therefore, increased, since the problem is more precisely defined.

These three variables offer a rubric under which specific research projects can be designed. Let us return to the problem of continuous control responses to illustrate further the applicability of this rubric.

Non-textual algorithms. Earlier we stated that the procedure for executing the vertical S-A described on p. 3 in this paper was not an algorithm. Let us examine the syntactic, semantic, and pragmatic aspects of this procedure to determine why this is so. Certainly the syntactic aspects of the procedure are sufficiently elementary for the intended class of users (student pilots). However, looking at the first cue in this procedure, we can see the semantic aspects of the procedure, as well as the pragmatic aspects, are not sufficiently elementary. The first cue requires the student pilot to move the throttle smoothly, slowly, and steadily. The semantic and pragmatic aspects of this prescription are not sufficiently elementary, since not all learners will interpret and respond to this cue in the same manner. There seems little else that can be done. How do you state unambiguously, in a few words, what kind of throttle movement is desired?

But algorithms need not be restricted to verbal (in this case, textual) semantic elements. Incongruous as it may sound, the semantic elements of an algorithm may be pictorial. Recall the algorithm for identifying the family to which an insect of the order neuroptera

belongs (p. 30): since it would be virtually impossible to describe all the defining characteristics textually, the algorithm includes pictures.

Carry this a step farther. Assume for the sake of discussion that a student pilot who can learn to distinguish between examples and non-examples of a "smooth, slow, and steady" throttle movement will be able to produce such a movement. The simulator is programmed in such a manner that the student pilot can put his hand on a moving throttle and "feel" what smooth, slow, and steady is as his hand moves along with the throttle. In effect, an algorithm might be constructed which would include, as one of its semantic aspects, a discriminator based on this kind of stimulus.

The application of algorithms to flying training could be very productive if methods of representing the "semantic" aspects can be developed. This kind of activity is a potentially high-yield endeavor. Many aspects of flying demand responses to such stimuli as curves, moving objects, and other similar types of analog information. Ultimately, the kind of research activity suggested should lead to better methods of dealing with analog information--a very critical aspect of learning to fly.

* * *

The concept algorithm represents a potentially powerful variable for learning and teaching. As we learn more and more about algorithms, prescriptions for the instructional systems designer should begin to emerge. At present, the problem is not so much one of finding a researchable problem under the rubric algorithm as it is selecting the best of many candidates for the first mission.

EPILOGUE

There is, of course, the possibility that algorithms are "no damn good." Conceivably, research might demonstrate that they do not facilitate learning, teaching, or instructional design. What then?

We have an algorithm to cover that eventuality, too (Snoopy, 1975). "Go back to Section I. Reread the story of Houdini. Then try magic."

References

Bellman, R., Cooke, K. L., and Lockett, J. A. Algorithms, graphs and computers. New York: Academic Press, 1970.

Brecke, F. H., Gerlach, V. S., and Shipley, B. D. Effects of instructional cues on complex skill learning. (Technical Report No. 40829, Project No. AFOSR 71-2128) Arlington, VA: U. S. Air Force Office of Scientific Research, 1974.

Bung, Klaus. A simplified notation for Ljapunov algorithms and their meta-algorithm. Unpublished mimeographed paper, 1969.

Bung, Klaus. A cybernetic approach to programmed language instruction. Educational media international, 1971, 4, 1-8.

Bussmann, H. Zur Kybernetik des Lernprozesses. Duesseldorf: Paedagogischer Verlag Schwann, 1971.

Davies, I. K. "Selecting an appropriate strategy for communication complex rules, procedures and instructions." The Management of Learning, Ch. 9. New York: McGraw-Hill, 1971.

Doctorow, E. L. Ragtime. New York: Random House, 1975.

Ehrenpreis, W., and Scandura, J. M. An algorithmic approach to curriculum construction in mathematics: A field test. Structural learning series, Report 64. Philadelphia: University of Pennsylvania, March 1972.

Eubanks, J. Rule learning and the design of systematic training. (Technical Report No. 60115, Project No. AFOSR 75-2900) Bolling AFB, DC: U. S. Office of Scientific Research, 1976.

Frank, H. Ueber die Kalkuelisierbarkeit der didaktischen Variablen von Paul Heimann. In: Northemann, W., and Otto, G. (Eds.), Geplante Information. Weinheim: Verlag Julius Beltz, 1969.

Glushkow, V. M. Theorie der abstrakten Automaten. (G. Asser, Ed., and trans., K. Straehmel, trans.) Berlin: Deutscher Verlag der Wissenschaften, 1963.

Horabin, I., and Lewis, B. Algorithms. Charles Town, WV: Ivan Horabin, 1974.

Horst, D. P., Talmadge, G. K., and Wood, C. T. A practical guide to measuring project impact on student achievement. Washington, DC: U. S. Department of Health, Education and Welfare, National Institute of Education, 1975.

Katona, G. Organizing and memorizing. New York: Columbia University Press, 1940.

Knuth, D. E. The art of computer programming, Vol. 1, Fundamental algorithms. Reading, MA: Addison-Wesley, 1968.

Landa, L. N. Algorithmization in learning and instruction. Englewood Cliffs, NJ: Educational Technology Publications, 1974.

Lansky, M. Learning algorithms as a teaching aid. RECALL: Review of educational cybernetics and applied linguistics, 1969, 1, 81-89.

Lyapunov, A. A. The logical structure of programmes. In: A. A. Lyapunov (ed.): Problems of cybernetics 1. Oxford: Pergamon Press, 1960.

Markov, A. A. Theory of algorithms. Washington, DC: National Science Foundation, 1961.

Scandura, J. M. Deterministic theorizing in structural learning: Three levels of empiricism. Journal of structural learning, 1971, 3 (1), 21-53.

Trakhtenbrodt, B. A. Algorithms and automatic computing machines. Lexington, MA: D. C. Heath, 1965.

Weaver, W. Recent contributions to the mathematical theory of communication. In: Shannon, C. E., and Weaver, W. The mathematical theory of communication. Urbana, IL: The University of Illinois Press, 1948.

APPENDIX A

A SHORT HISTORY OF ALGORITHMS

Manhind has used algorithms or algorithmic-like prescriptions for at least as long as it has engaged in such sociological phenomena as the division of labor or the establishment of laws, rules and procedures. In the context of social control the concept of an algorithm is understood as a general procedure for the solution of a specific class of problems.

Formal algorithms probably appeared first in the field of mathematics. The prime example is the Euclidean Algorithm. Another example from antiquity is Aristotle's (384-322 B.C.) Syllogistics in which he set forth a system of rules dealing with certain forms of logical conclusions. More general applications of algorithms as universal computation procedures in mathematics did not appear until the Middle Ages.

Our current decimal system originated in India and was adapted by the Arabs around the mi-dle of the 7th Century. In one of his works on mathematical and astronomical problems, Muhammad Ibn Musa Al-Hwarizmi (ca. 825 A.D.) explains the Indian (or Arabic) number system as well as computational procedures within this system. The original manuscript is lost, but according to one theory the algorithm was taken directly from the title of the Latin translation: "Algorithmi de numeri Indorum . . ." Another theory suggests that the word was derived from the name of the author, Al Hwarizmi.

The algebraic methods introduced by Arabian mathematicians greatly influenced Arymondo Lullus (ca. 1300 A.D.). Lullus was the author of Ars Magna, which he considered to be a general procedure for discovering all truths. The procedures or methods which he actually supplies are of little or no practical value. His genius lies in the conception of the idea of a general method which exerted a very strong influence on following generations of mathematicians. For example, Cardano, more than 200 years later, still conceived of algebra as "the art of Lullus." Evidence to this effect is found, among other sources, in his Artis Magna Seu de Regulis Algebraicis (1545), in which he published algebraic algorithms, including the algorithm for the solution of tertiary equations which was named after him.

During the 16th century most writers in the field of algebra apparently believed that all algebraic problems could be treated algorithmically. This was not the case with geometry, the only other branch of mathematical science then in existence. Descartes (1596-1650), the father of analytical geometry, attempted to develop a method of translating all problems of geometry into algebraic form(s), thus rendering them amenable to solution by means of algebraic algorithms. In his view, all algebraic problems could be solved by applying one or more algo rithms; consequently, he concluded that there were no interesting problems remaining for a creative mathematician. Considering the state of the art at this time, this was a pardonable error. Almost three hundred years passed before it was possible to delimit the range of mathematical problems that could be solved algorithmically.

The concept of an Ars Magna led Leibniz (1646-1716) to attempt to develop algorithms which were as general as possible. Leibniz emphasized that his deliberations were based on Lullus and that the concept of an Ars Magna actually encompassed two component concepts, namely that of an ars judicandi (decision procedures) and that of an ars inveniendi (production procedures). He also pointed out that a truly algorithmic procedure must be executable by a machine, a mechanism. He was far ahead of his time in that he was describing, in a primitive way, an information processing machine which, of course, is one of the basic concepts of modern cybernetics.

Following Leibniz, little or no effort was made to develop the concepts of Ars Magna because of a lack of suitable methodologies for mathematical formulation and interpretation. Not until the 19th Century where De Morgan, Boole, and Schroeder began to develop such methods in the field of logic, did this situation begin to change.

The desire for an exact algorithmic basis for mathematics, which evolved from set theory, led to new directions in the formalization of logic as well as mathematics. These investigations, which dispensed with a close analogy to algebra, were begun by Frege and Peano and they culminated in Whitehead's and Russell's monumental work Principia Mathematica (1901-1908), in which it was demonstrated that much of logic and mathematics could be represented in the form of a "calculus" (a formalized language or formalized theory). Finally, the work of D. Hilbert and his students as well as that of the Polish School of Logicians, who in the early 1900's contributed fundamental works on the development and structure of formalized theories, deserves mention.

Two essential factors contributed to the adaptation of the algorithmic concept by the behavioral sciences. The first of these was the advent of modern computer technology. The hardware suggested models concerned with aspects of the functioning of the human mind; the software (i.e., the programs themselves which are algorithms) led to the more involved, precise, and creative activity of artificial intelligence modeling. The description and modeling of problem solutions and isomorphic mental solution processes by means of algorithms offers the potential of a very high degree of precision in terms of both analysis and representation. An excellent example of this type of development is found in the work of Newell and Simon (1972).

The second major contributing factor was the emergence of cybernetic theory, which supplied the theoretical framework as well as the mathematical apparatus for a synthesis of the two branches of artifical intelligence modeling. N. Wiener (1948) supplied the fundamental theory for general cybernetics; Glushkov (1966) contributed major extensions to automation theory. Among the first writers to apply concepts of cybernetic theory and of algorithmic theory to educational problems were Frank (1962; 1969) and Landa (1966). The latter wrote Algorithms in Learning and Instruction, which is the most significant and seminal work on the subject to date. The primary objective of his model is to train the learner in systematic methods of thinking; the method for achieving this objective is the use of algorithms.

The publication of Landa's book in the U. S. in 1974 is part of an increasing tendency in this country to treat basic issues in learning and instruction in terms of theoretical constructs which are deduced from a cognitive-cybernetic model rather than from a purely behavioristic model of psychology. This shift in orientation is accompanied by the emergence of a new vocabulary, of which "algorithm" is but one word. The term and the concept, however, are now as much a part of the educator's vocabulary as are "objectives" or "criterion referenced testing." Yet, while "objectives" and "criterion referenced testing" are notions which are very clearly defined and delimited in meaning an usage, "algorithms" still represents a term which is in need of a precise definition. This report attempts to accomplish just that.